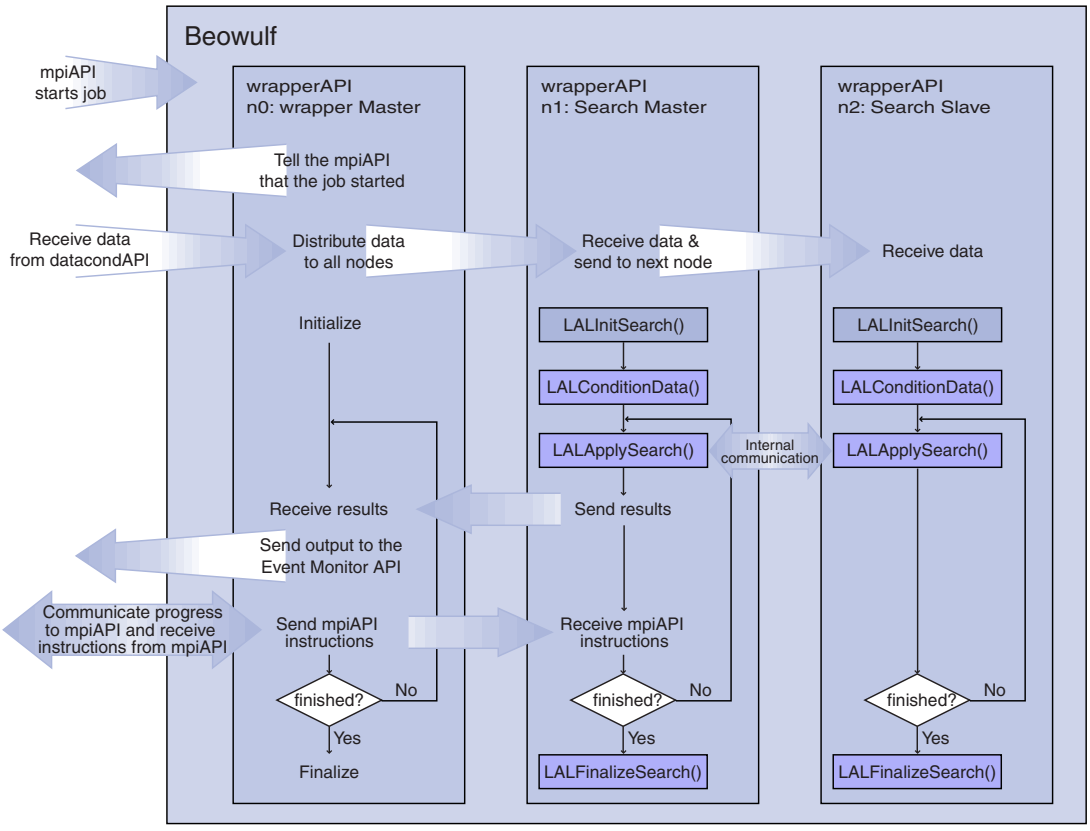
LALWrapper Software Documentation



RCS Revision: 1.24 Date: 2003/10/08 15:05:12 UTC — Compiled: 2003/10/09

Contributors

Anderson, Warren G
Brady, Patrick R
Brown, Duncan A
Charlton, P R
Creighton, Jolien D E
Creighton, Teviet
Daw, Ed
Edlund, Jeff
Fang, Hua
Heng, Siong
Mendell, Greg
Rahkola, Rauha
Reilly, Kaice T
Romano, Joseph D
Sylvestre, Julien
Tibbits, M M
Tinto, Massimo
Torres, Charlie W
Wen, Linqing
Whelan, John T
Williamsen, Mark S
Wiseman, Alan G
Yakushin, Igor

Contents

Preface

This manual describes the generic LAL-Wrapper interface and documents the various contributed shared objects.

0.1 Introduction

The preferred method of developing code to be executed by the `wrapperAPI` is to install LAL, LALwrapper and a standalone `wrapperAPI` on your system and your shared object within the LALwrapper source tree. This allows the configure script to generate the necessary makefiles with the appropriate flags.

Before we begin with the technical aspects of development, a brief reminder of the software model adopted by the LSC may be useful. There are three software components: (i) the LIGO data analysis system (LDAS), (ii) the LIGO/LSC Algorithm Library (LAL) and (iii) the LAL-LDAS interface and search code library (LALwrapper).

0.1.1 LDAS

This is the exoskeleton which supports the internal organs of the LSC search codes. It is developed and maintained by the LIGO Laboratory. It is broken into modules – the technical term for these modules is an Application Programmer Interface (API) – which provide different pieces of functionality required by a search pipeline. For example, the frame API can read frames from disk or archive, extract information requested by the user, concatenate the requisite time series as necessary and send the information to the next API. In general, the LDAS API's adopt a model in which they receive data through an inflow (a data socket), apply methods to the data internally, and then pass the data to the next API through some outflow (another data socket). This model means that the various API's can be assembled into many different topologies which allow complex and difficult data analysis tasks to be carried out. There is a standard method for logging information from each API and a special API which allows the user to monitor and control jobs (the Control Monitor API). Thus, LDAS provides a standardized infrastructure for accessing and manipulating the data, and for keeping detailed records of the analysis. Scientific search codes slot into LDAS via the wrapperAPI [1, 2]. The search codes are *not* part of LDAS software.

0.1.2 LAL

This library contains the numerical algorithms used to construct the gravitational wave search codes. It is developed and maintained by the LSC. The LAL specification determines the data types, calling format, and other coding requirements for functions in this library. All scientific searches will be executed using this library [3]. The functions in LAL are like cells. Each function can do something (relatively) simple. When many functions are put together in an appropriate manner, they can execute complex activities like searching for waves from coalescing compact binaries.

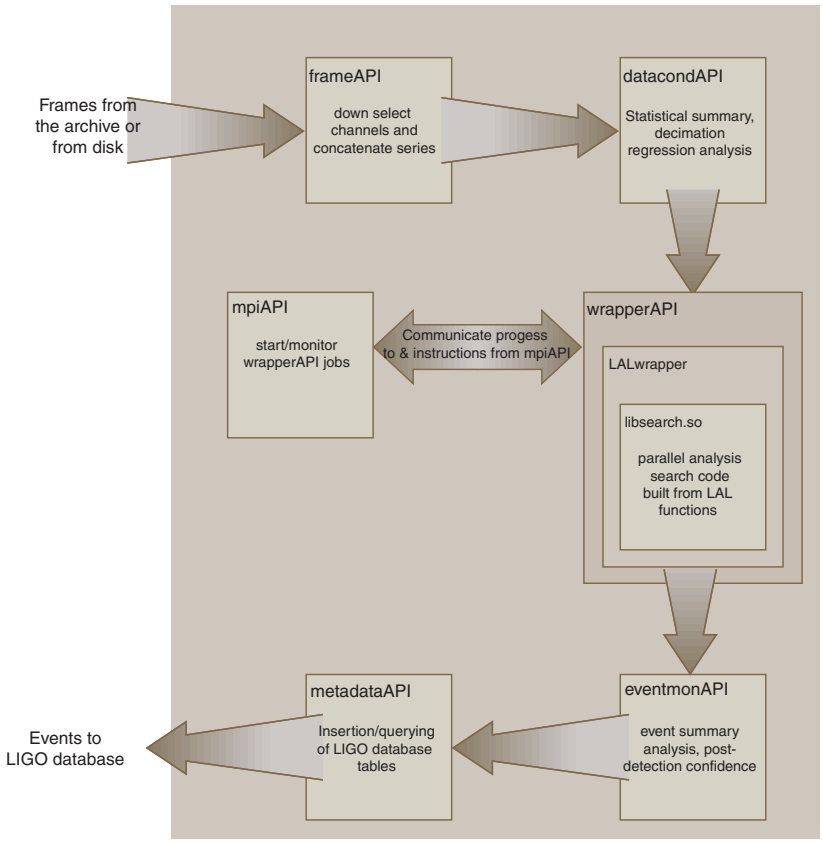


Figure 1: This figure illustrates a simple example data flow through LDAS. LALwrapper and LAL are shown inside the wrapperAPI.

0.1.3 LALwrapper

This is a set of libraries. It is written and maintained by the LSC. The LAL specification governs coding practice in the search codes implemented through this package [3]. A library might execute a type of search, e.g. a search for waves from coalescing compact binaries, or some other computationally intensive data manipulation. Each library contains five functions which are called by the wrapperAPI: `initSearch()`, `conditionData()`, `applySearch()`, `freeOutput()`, and `finalizeSearch()`. Developers do not deal directly with these five functions, instead they deal with four LAL compliant counterparts: `LALInitSearch()`, `LALConditionData()`, `LALApplySearch()`, and `LALFinalizeSearch()`. The `freeOutput()` function is the same in all search codes; it is needed only because wrapperAPI cannot free memory allocated using `LALMalloc()`. A detailed specification of these functions and their requirements can be found in [1]. `LALApplySearch()` is called in a loop on all search nodes (see Fig. 2). It executes the search algorithm. A few points about `LALApplySearch()`:

- The MPI communicator is passed to this function. This allows for truly parallel implementations of search algorithms to be coded libraries since MPI processes on any of the search nodes can, in principle, communicate with others.
- This function must return at least 10 times during a search, see Ref. [1]. On the search master node, it may return more often. For example, in a slave driven search code it might return each time that a slave node sends results to the search master.

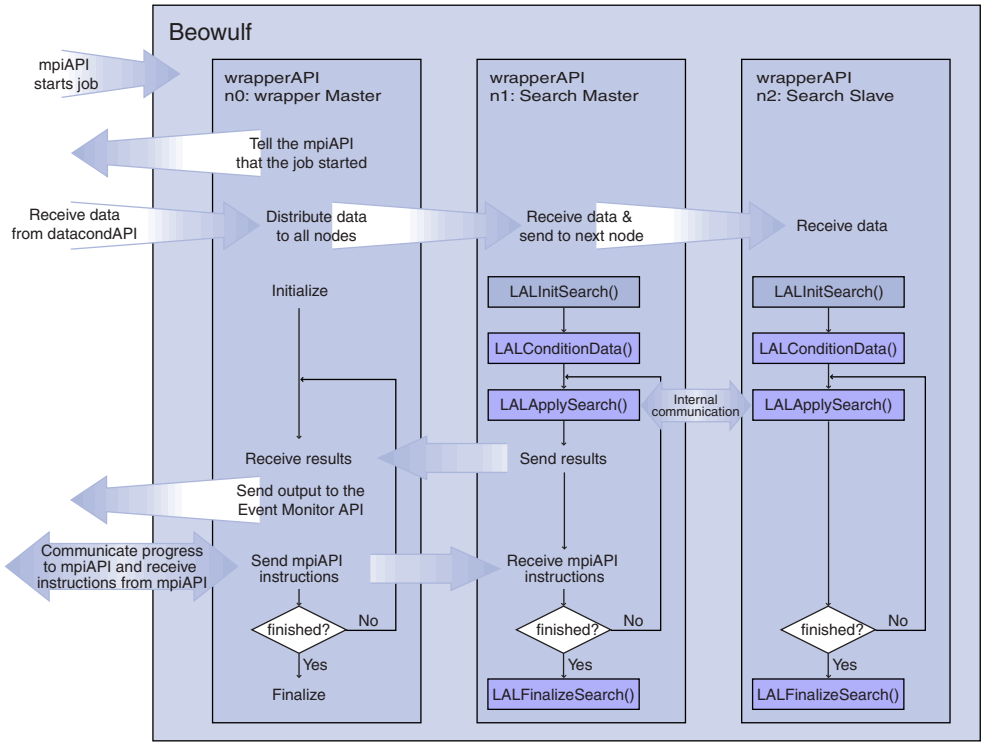


Figure 2: This figure illustrates the logic of the wrapperAPI with respect to the four search specific functions loaded from the LALwrapper shared object library. This wrapperAPI job uses three nodes: n0, n1, n2. If the job used more than this, the logic and communication of nodes n3–nN would be the same as n2. Note the naming convention used above. The *wrapper master* is always n0; it is responsible for broadcasting the data (this is the default behavior, see the wrapperAPI baseline requirements for other options), sending/receiving job information to/from the mpiAPI, buffering results from the search code and sending them to the eventmonAPI. Scientific search code is not executed on the wrapper master. The search is executed on nodes n1–nN. The *search master* is always n1. It is the master process associated with any parallel search algorithm. The *search slaves* are nodes n2–nN.

- Upon return, the boolean flag `output->notFinished` must be set; this must be done on all nodes not just the search master. `FALSE` indicates that applysearch is finished on that node. On the search master, `output->notFinished` should not be set to `FALSE` until all search slave nodes have finished. In addition, on the search master `output->fracRemaining` must be set to a number between 0 and 1 when `LALApplySearch()` returns. This number indicates the progress of the job by informing the wrapperAPI what fraction of the job remains to be executed. This information is used by the wrapperAPI and mpiAPI to track the progress of jobs, so it should be estimated as accurately as possible. (N.B. The wrapperAPI can be run in a *load balancing* configuration which requires further constraints on the execution `output->fracRemaining` and coordination on the search nodes. See Refs. [1, 2] for details.)

Part I

How to get started

Chapter 1

How to develop search code to run in LDAS under wrapperAPI

1.1 Getting and installing the software

We describe the installation of LAL, LALWrapper and the components of LDAS needed to run the wrapperAPI in standalone mode. Installing the software needed by LDAS takes a little time and effort. Once the infrastructure is in place, however, it is straightforward to develop search code to run under the wrapperAPI. Ultimately, testing should be done on a system running LDAS but a standalone wrapperAPI implementation is preferable during the first stage of development.

1.1.1 Installation of required development tools

LAL, LALWrapper and LDAS development takes place in a standardized environment of software tools. This avoids confusion caused by building with different tools and in different environments. The LDAS/LAL development tools are required to reside in a directory hierarchy available at /ldcg. (This directory may be a link to another, somewhere else on your system; however, these packages *must* be available through this reference.)

Detailed instructions describing the packages required for the stand alone wrapperAPI in /ldcg, their version numbers, where to obtain them, how to patch them (if necessary), and how to install them is maintained on the LDAS web page. Go to and follow the instructions in Sec. 1 of http://ldas-sw.ligo.caltech.edu/doc/INSTALL_WRAPPER.html to install the packages. *Follow only the instructions in Sec. 1; do not retrieve the source as described in Sec. 2.* You will need root permission on your system to carry out this installation process. Be sure to change the environment variables as explained in the LDAS install document.

1.1.2 Configuring your environment

While developing, we recommend that you install the LAL, LALWrapper and wrapperAPI somewhere under your home directory. If you follow the instructions below, the LAL, LALwrapper and LDAS 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.

You will need to choose a convenient location for the wrapperAPI resource file. It is acceptable to put it in the LALPREFIX directory. If you put it somewhere else, edit the WRAPPER_RESOURCE_FILE environment variable below.

To make the appropriate binaries accesible 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 or lalwrapper 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:/ldcg/bin:${PATH}
export WRAPPER_RESOURCE_FILE=${LALPREFIX}/share
export LAMBHOST=${LALPREFIX}/share/lam-bhost.def
export LALCVS=":pserver:anonymous@gravity.phys.uwm.edu:/usr/local/cvs/lal"
export LALWRAPPERCVS=":pserver:anonymous@gravity.phys.uwm.edu:/usr/local/cvs/lalwrapper"
export LDASCVS=":pserver:readonly@ldas-sw.ligo.caltech.edu:/ldcg_server/common/repository"
```

If you are using csh or a derivative, add the following lines to your .cshrc file

```
setenv LALPREFIX $HOME
setenv PATH $LALPREFIX/bin:/ldcg/bin:$PATH
setenv WRAPPER_RESOURCE_FILE ${LALPREFIX}/share
setenv LAMBHOST ${LALPREFIX}/share/lam-bhost.def
setenv LALCVS ":pserver:anonymous@gravity.phys.uwm.edu:/usr/local/cvs/lal"
setenv LALWRAPPERCVS ":pserver:anonymous@gravity.phys.uwm.edu:/usr/local/cvs/lalwrapper"
setenv LDASCVS ":pserver:readonly@ldas-sw.ligo.caltech.edu:/ldcg_server/common/repository"
```

These enviromnent variables must be set before running lamboot, 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 shared objects 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 $LALPREFIX
```

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

```
mkdir $LALPREFIX/src
```

The LAL software is maintained in a CVS repository – CVS stands for Concurrent Version-control System which is tool to allow multiple developers to manipulate the same software, meerging 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 a 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.

To build and install the LAL software suite,

```
cd $LALPREFIX/src/lal
./00boot
./configure --prefix=$LALPREFIX --disable-static --enable-mpi \
    --with-extra-cppflags="-I/ldcg/include" \
    --with-extra-cflags="-fexceptions" \
    --with-extra-ldflags="-L/ldcg/lib"
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 LALwrapper

Obtain the LALwrapper package from the the CVS repository as follows:

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

The password for the `anonymous` user is `lalwrapper`. The version of LALWrapper 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 $LALWRAPPERCVS checkout -rHEAD lalwrapper
```

You have now obtained the latest development version of LALWrapper.

To build and install the LALWrapper software suite,

```
cd $LALPREFIX/src/lalwrapper
./00boot
./configure --prefix=$LALPREFIX \
    --with-extra-cppflags="-I$LALPREFIX/include -I/ldcg/include" \
    --with-extra-ldflags="-L$LALPREFIX/lib -L/ldcg/lib"
make
make check
make dvi
make install prefix=$LALPREFIX/stow_pkgs/lalwrapper-howto
cd $LALPREFIX/stow_pkgs
stow lalwrapper-howto
```

This completes the installation of LALwrapper.

1.1.5 wrapperAPI

To build the `wrapperAPI` in standalone mode requires a subset of LDAS components to be downloaded. The easiest way to get these is through the CVS repository. There is a `readonly` user; contact Kent Blackburn or Albert Lazzerini to obtain the password. The following commands also checkout the latest CVS version of the wrapperAPI and the components of LDAS that it needs. The version of LDAS 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 `ldas-X_Y_Z` where `X`, `Y`, `Z` are integers which identify the release version number as `X.Y.Z`.

Log into the repository and download the necessary files as follows:

```
cd $LALPREFIX/src
cvs -d $LDASCVS login
cvs -d $LDASCVS checkout -rHEAD -l ldas
cvs -d $LDASCVS checkout -rHEAD -l ldas/lib
cvs -d $LDASCVS checkout -rHEAD ldas/lib/perceps
cvs -d $LDASCVS checkout -rHEAD ldas/lib/general
cvs -d $LDASCVS checkout -rHEAD ldas/lib/ilwd
cvs -d $LDASCVS checkout -rHEAD ldas/lib/dbaccess
cvs -d $LDASCVS checkout -rHEAD ldas/dbms
cvs -d $LDASCVS checkout -rHEAD -l ldas/api
cvs -d $LDASCVS checkout -rHEAD ldas/api/wrapperAPI
cvs -d $LDASCVS checkout -rHEAD ldas/api/genericAPI
cvs -d $LDASCVS checkout -rHEAD ldas/api/test
```

Note that some commands involve the `-l` option and others do not. The distinction is important: do not lose it.

To build LDAS, issue the commands

```
cd $LALPREFIX/src/ldas
./build-ldas --prefix=$LALPREFIX --disable-metadata-api
```

The `build-ldas` command will take some time to complete (on order 10m on an unloaded, relatively modern system). When the build starts, a subdirectory will be created into which all the programs, libraries and log files are put. The directory name follows the particular system on which you compile, e.g. on a Pentium III running Linux, the directory would be called `Linux-i686`. For simplicity, I use this name below but it may be different on your system.

You can check the progress of the build as follows. Using another terminal window, `cd` into the directory `$LALPREFIX/src/ldas/Linux-i686/build_logs`. This directory will contain several files which are created as different phases of the build proceed. If the build completes, there should be no error messages in the file `make`.

Once the build completes, finish the installation process as follows (remember to replace `Linux-i686` with the appropriate directory name):

```
cd $LALPREFIX/src/ldas/Linux-i686
make install
cd $LALPREFIX/stow_pkgs
stow ldas-X.Y.Z
```

where `X`, `Y`, `Z` are integers which identify the version number.

Now copy the LDASwrapper.rsc resource file to the location specified in the `WRAPPER_RESOURCE_FILE` enviroment variable.

```
cp $LALPREFIX/bin/LDASwrapper.rsc $WRAPPER_RESOURCE_FILE
```

Edit the LDASwrapper.rsc resource file as follows:

1. To disable communication with the mpiAPI, change the `enable_mpiAPI` from `TRUE` to `FALSE`.
2. To disable communication with the resultAPI, change the `enable_resultAPI` lines from `TRUE` to `FALSE`.
3. `wrapperAPI` uses `dump_data_directory` (default value is `/ldas_outgoing/jobs`) when `enable_resultAPI` is set to `FALSE`. Please change `dump_data_directory` from `/ldas_outgoing/jobs` to `/tmp/ldas_outgoing/jobs`.
4. `wrapperAPI` uses `run_code` to determine some of the directory names under `dump_data_directory`. Change the `run_code` from `LDAS-DEV` to `NORMAL`.

5. The actual directory where files will be written is defined by: `$(dump_data_dir)/$(run_code)_N/$(run_code)jobID`, where integer N = jobID / 10000; i.e. using default values for the resource variables data for job with ID=8 would be written to the `/ldas_outgoing/jobs/NORMAL_0/NORMAL8` directory. Create these directories with:

   ```
   mkdir -p /tmp/ldas_outgoing/jobs/NORMAL_0/NORMAL8
   ```

   Create the corresponding directory.

This completes the installation of the `wrapperAPI`.

1.2 Running your first wrapperAPI job

Now that you have installed the software required to run search code under the wrapperAPI in standalone mode, you want to test everything together. The following steps lead you through your first parallel execution using the wrapperAPI. First, check that you set LAMBHOST environment variable as described above. Once it is set, create a boot schema file for LAM:

```
echo localhost > $LAMBHOST
```

Then, go to the example subdirectory of the lalwrapper source code and start the LAM daemon:

```
cd $LALPREFIX/src/lalwrapper/example
lamboot -v
```

The output should read something like

```
LAM 6.5.2/MPI 2 C++ - University of Notre Dame

Executing hboot on n0 (naoise.phys.uwm.edu)...
topology done
```

with your computer name replacing naoise.phys.uwm.edu. The version of LAM should match that which you have installed in /ldcg by following the instructions on the LDAS web pages.

Next, we run the wrapperAPI using the trivial shared object provided in LALwrapper:

```
mpirun -v trivial.schema
```

The verses of *Swinging on a Star* are printed to stdout. Congratulations, you have just run your first wrapperAPI job. The next section contains reference information about the file trivial.schema and the input file that must be provided to the wrapperAPI even if the data is ignored by the LALwrapper shared object.

Important Note: LAL, LALwrapper and the wrapperAPI are interdependent. Changes to LAL often require changes to LALwrapper if the latter is to successfully compile. Similarly, some changes to the wrapperAPI require changes to LALWrapper. As a general rule, you should alway check-out the same version of LALwrapper as you have of LAL. In particular, if you are working with the development snapshot of LAL, you will almost certainly also have to be working with the development snapshot of the other. If the trivial wrapperAPI fails to run, check that LAL and LALWrapper are compatable – a quick review of recent postings to the lal-discuss mail archive will help here. If so, and you still have problems, send an e-mail to mpiteam@gravity.phys.uwm.edu. We will reply as quickly as possible.

1.2.1 LAM schema files

There are two files needed by this test. The first is trivial.schema, this is a schema file used by LAM to determine how to execute parallel MPI code. The file contains a single line which is not commented. Since this line is too long to display here without breaking it up, we suggest that you open the file with an editor while you read the explanation of the different parts of the line. Here are the arguments and what they refer to:

h LAM argument to specify only run on node where mpirun call is made

-np 4 Start up the program (wrapperAPI) as 4 parallel processes. Remember n0 is the wrapper master and does no search specific processing, n1 is the search master and n2–n3 are search slaves.

wrapperAPI The program LAM runs

-mpiAPI="(marfik.ligo.caltech.edu,10000)" The socket address to connect to the mpiAPI; this is ignored by the standalone wrapperAPI.

-nodelist="(1-3)" The wrapperAPI only executes the LALwrapper library routines on nodes 1–3.

-dynlib="/home/patrick/lib/lalwrapper/libldastrivial.so" The location of the trivial shared object that was installed with the lalwrapper software.

-dataAPI="(datahost,5678)" The socket address where the datacondAPI is running; this is ignored by the standalone wrapperAPI. Instead data is read in from a file (in ILWD format) which is specified in a later argument.

-resultAPI"=(reshost, 9101)" The socket address to connect to the Event Monitor API; this is ignored in standalone mode and the output is written to a file.

-filterparams="(1,0)" A comma separated list of parameters needed by the shared object to run the search code.

-realTimeRatio=0.9 The time for execution of the search must be less than 0.9 length of input time series in seconds.

-doLoadBalance=FALSE Whether or not to do dynamical load balancing. Should always be false when runnning standalone, since the mpiAPI is required to execute load balancing.

-dataDistributor=W The **W**rapperAPI is responsible for distributing the data. Other options are described in Ref. [2].

-jobID=8 A unique job ID; for standalone execution, this must agree with teh jobID in the input-File below. It is handled by the mpiAPI under normal conditions.

-inputFile="wrapper.ilwd" The location of the file from which data should be read. The data must be in a consistent ILWD format. See the wrapperAPI baseline requirements for examples.

1.2.2 wrapperAPI input files, e.g. wrapper.ilwd

The input files read from disk must be in ILWD format. (The file for this example is called wrapper.ilwd, look at it in an editor.) The trivial example discussed above does not use the data, however the wrapperAPI requires an input file for execution. The structure of these files is described in detail in [2]; please refer to this document should you require additional information.

1.2.3 wrapperAPI output files

When you run the wrapperAPI in standalone mode as described so far, it will generate one or more output files depending on the LALWrapper shared object it loads:

- process_n.ilwd These files contain process information that will be sent to the database when run as part of an LDAS pipeline.
- output_n.ilwd These files contain output that was generated by the LALWrapper shared object when it was executed. The number of files and their content will depend on the shared object and the input data.

All other information will be printed to stderr and stdout; note, however, this mode of output should only be generated when the code is compiled and run with debugging turned on.

Once you want to test code on real data, you will move to running your job on an instantiation of LDAS which can read frame data and pass it through full pipelines. An explanation of how to do that will be provided in the future. (Target date for such a HOWTO is 21 May 2001.)

1.3 Adding helloworld to LALwrapper

This example explains how to add a shared object called helloworld to LALwrapper.

1. If you have just configured and compiled LALwrapper, then you need to clean up before proceeding to the next step

   ```
   cd $LALPREFIX/src/lalwrapper
   make cvs-clean
   ```

2. Create a directory helloworld in the contrib subdirectory of your lalwrapper source distribution:

   ```
   cd $LALPREFIX/src/lalwrapper/contrib
   mkdir helloworld
   ```

3. The helloworld directory should contain subdirectories src, include, doc, and test and a Makefile.am that contains the single line:

   ```
   SUBDIRS = include src test doc
   ```

   Execute the following commands

   ```
   cd helloworld/
   mkdir include src test doc
   echo "SUBDIRS = include src test doc" > Makefile.am
   ```

4. Enter the include directory and copy include/Makefile.am from contrib/trivial

   ```
   cd include
   cp ../../trivial/include/Makefile.am .
   ```

   Edit this Makefile.am and change the line

   ```
   noinst_HEADERS = Trivial.h
   ```

   to list all the header files in your package, for what follows this will read:

   ```
   noinst_HEADERS = HelloWorld.h
   ```

   The header file follows the LAL conventions. An example can be found in Sec. 1.3.1; add this content to a file called HelloWorld.h in this subdirectory.

5. Enter the src directory and copy src/Makefile.am from contrib/trivial

   ```
   cd ../src
   cp ../../trivial/src/Makefile.am .
   ```

   In this Makefile.am change every "libldastrivial" to "libldashelloworld". Change

   ```
   pkglib_LTLIBRARIES = libldastrivial.la
   ```

   to

   ```
   pkglib_LTLIBRARIES = libldashelloworld.la
   ```

change

```
libldastrivial_la_SOURCES = Trivial.c TrivialSong.h
```

to list your source files, for example

```
libldashelloworld_la_SOURCES = HelloWorld.c
```

and change

```
libldastrivial_la_LIBADD = $(top_builddir)/src/liblalwrapper.la
```

to

```
libldashelloworld_la_LIBADD = $(top_builddir)/src/liblalwrapper.la
```

Because the library is so simple, the single file can contain all four functions. An example can be found in Sec. 1.3.2; add this content to a file called `HelloWorld.c` in this subdirectory.

6. Enter the test directory and copy test/Makefile.am from contrib/trivial

```
cd ../test
cp ../../trivial/test/Makefile.am .
```

Change the line

```
noinst_SCRIPTS = TrivialTest.sh
```

to list the names of the test scripts you wish to run, for example

```
noinst_SCRIPTS = HelloWorld.sh
```

These test scripts should use the `happyAPI` to exercise the shared object. Note that the happyAPI cannot handle complicated input and output. Only the simplest shared objects can be tested this way. Now, copy `TrivialTest.sh`

```
cp ../../trivial/test/TrivialTest.sh ./HelloWorld.sh
```

and edit the line

```
args=../src/.libs/libldastrivial.so
```

to read

```
args=../src/.libs/libldashelloworld.so
```

Finally, insure that the `HelloWorld.sh` is executible

```
chmod +x HelloWorld.sh
```

7. Enter the doc directory and copy `doc/Makefile.am` from `contrib/trivial`

```
cd ../doc
cp ../../trivial/doc/Makefile.am .
```

Edit this Makefile.am and change the line

```
PACKAGE = trivial
```

to

```
PACKAGE = helloworld
```

The makefile in the doc directory uses the program laldoc to extract documentation from the .c and .h files. There should be two texfiles in the doc subdirectory:

main.tex: this file is used to make the documentation in the contrib/helloworld/doc directory

helloworld.tex: this file is read by main.tex to make the documentation in the contrib/helloworld/doc directory and is also read by lalwrapper.tex in the top-level doc directory to make the helloWorld documentation in the lalwrapper.pdf

The main.tex file is almost the same in all directories. First, copy the main.tex from the trivial package

```
cp ../../trivial/doc/main.tex .
```

then edit the file and replace \include{trivial} with \include{helloworld}.

The helloworld.tex file does three things: (i) it begins a new chapter of documentation for the helloworld package, (ii) it contains general information not extracted from the source or header file, and (iii) it inputs the documentation for each module that is contained in the package.

An example can be found in Sec. 1.3.3; add this content to a file called helloworld.tex in this subdirectory.

Finally, any figures should be included in the doc directory. These figures should begin with the package name, e.g., helloworldFig1.eps, helloworldFig1a.eps, helloworldPicture.eps, etc. Both eps and pdf versions of the figures should be present.

8. Go up to the contrib directory and edit the Makefile.am there

```
cd ../..
vi Makefile.am
```

Add your package to the "SUBDIRS =" line:

```
SUBDIRS = trivial sick load inspiral power helloworld
```

9. Go to the top level and edit the file configure.in

```
cd ..
vi configure.in
```

At the bottom of this file is a large section that begins

```
AC_OUTPUT( Makefile                                    \
```

At the end of this section, before the closing brace, add the Makefiles to be generated in your package. To do this, just add the lines

```
contrib/helloworld/Makefile          \
contrib/helloworld/doc/Makefile      \
contrib/helloworld/include/Makefile  \
contrib/helloworld/src/Makefile      \
contrib/helloworld/test/Makefile     \
```

10. In `$LALPREFIX/src/lalwrapper` we now recompile `lalwrapper`:

```
cd $LALPREFIX/src/lalwrapper
./00boot
./configure --prefix=$LALPREFIX \
    --with-extra-cppflags="-I$LALPREFIX/include -I/ldcg/include" \
    --with-extra-ldflags="-L$LALPREFIX/lib -L/ldcg/lib"
make
make check
make dvi
make install prefix=$LALPREFIX/stow_pkgs/lalwrapper-new
cd $LALPREFIX/stow_pkgs
stow --delete lalwrapper-howto
stow lalwrapper-new
```

    This completes the process of adding and compiling your helloWorld library.

11. You can execute this code in the following way. Go to the `contrib/example` directory and make a LAM schema file to run the library:

```
cd $LALPREFIX/src/lalwrapper/example
cp trivial.schema hello.schema
```

    This file contains one long line which should not be broken, so beware is you are using an editor which autowraps. Change the occurence of `libldastrivial.so` to `libldashelloworld.so`, then

```
lamboot -v
mpirun -v hello.schema
```

    You should see the words `Hello, LSC!` written to stdout. Now you have run a wrapperAPI job using a shared object that you have written and added to the lalwrapper package yourself. Congratulations!

12. It's now time to start looking at the example search codes in the `contrib` directory. Either `power` or `inspiral` should serve as good examples to get started. Good luck and happy coding.

1.3.1 HelloWorld.h

```
/******** <lalVerbatim file="HelloWorldHV"> ********
Author: Brady, P R
$Id: HelloWorldH.tex,v 1.1 2001/04/26 15:24:21 patrick Exp $
********* </lalVerbatim> ********/

#ifndef _HELLOWORLD_H
#define _HELLOWORLD_H

#include <lal/LALRCSID.h>
#include <LALWrapperInterface.h>

NRCSID( HELLOWORLDH, "$Id: HelloWorldH.tex,v 1.1 2001/04/26 15:24:21 patrick Exp $");

/******************************** <lalErrTable file="HelloWorldHErrTab"> */
#define HELLOWORLDH_ENULL 1
#define HELLOWORLDH_MSGENULL "Null pointer."
/********************************** </lalErrTable> */

#endif /* _HELLOWORLD_H */
```

1.3.2 HelloWorld.c

```
/******** <lalVerbatim file="HelloWorldCV"> ********
Author: Brady, P R
$Id: HelloWorldC.tex,v 1.2 2001/12/06 23:19:00 patrick Exp $
********* </lalVerbatim> ********/
#include <stdio.h>
#include <lal/LALStdlib.h>
#include <lal/LALHello.h>
#include <HelloWorld.h>

NRCSID( HELLOWORLDC, "$Id: HelloWorldC.tex,v 1.2 2001/12/06 23:19:00 patrick Exp $");

// Initialization routine does nothing but checks its arguments
void LALInitSearch(
    LALStatus             *status,
    void                 **searchParams,
    LALInitSearchParams   *initSearchParams
    )
{
  INITSTATUS( status, "LALInitSearch", HELLOWORLDC );
  RETURN( status );
}

// ConditionData routine does nothing but checks its arguments
void
LALConditionData(
    LALStatus             *status,
    LALSearchInput        *inout,
    void                  *searchParams,
    LALMPIParams          *mpiParams
    )
```

```
{
  INITSTATUS( status, "LALConditionData", HELLOWORLDC );
  mpiParams=NULL;

  RETURN( status );
}

// Apply search is called once on search master and search slaves
// It prints a string to stdout using the LAL function LALHello()
void
LALApplySearch(
    LALStatus             *status,
    LALSearchOutput       *output,
    LALSearchInput        *input,
    LALApplySearchParams  *params
    )
{
  INITSTATUS( status, "LALApplySearch", HELLOWORLDC );
  ATTATCHSTATUSPTR (status);

  if ( !(output) ){
    ABORT( status, HELLOWORLDH_ENULL, HELLOWORLDH_MSGENULL);
  }

  LALHello(status->statusPtr,NULL);
  CHECKSTATUSPTR (status);

  output->numOutput     = 0;
  output->result        = NULL;
  output->fracRemaining = 0;
  output->notFinished   = 0;

  DETATCHSTATUSPTR (status);
  RETURN( status );
}

// Finalize routine does nothing but checks its arguments
void
LALFinalizeSearch(
    LALStatus             *status,
    void                 **searchParams
    )
{
  INITSTATUS( status, "LALFinalizeSearch", HELLOWORLDC );
  RETURN( status );
}
```

1.3.3 helloworld.tex

```
\chapter{Package: \texttt{helloWorld}}
This shared object prints does very little.
\newpage
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\section{Header \texttt{HelloWorld.h}}
```

```
\label{s:HelloWorld.h}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\noindent Provides routines to be executed by the wrapperAPI which prints
"Hello, LSC!" to stdout.

\subsection*{Synopsis}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\subsection*{Error Conditions}
\input{HelloWorldHErrTab}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\subsection*{Structures}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\subsubsection*{struct \texttt{MyStruct}}
\index{\texttt{MyStruct}}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\noindent Stuff about the structure ......

\begin{description}
\item[\texttt{TYPE element}] What it is .....
\end{description}

\vfill{\footnotesize\input{HelloWorldHV}}
\newpage
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\subsection{Module \texttt{HelloWorld.c}}
\label{ss:TimeSeriesToTFPlane.c}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Contains the four LAL functions neede to run search code .......

\subsubsection*{Prototypes}
\vspace{0.1in}
Autodoc your prototypes if there are any ..... see the LAL spec for how to do
this.

\subsubsection*{Description}

\subsubsection*{Uses}

\subsubsection*{Notes}

\vfill{\footnotesize\input{HelloWorldCV}}
```

References

[1] MPI working group, *LAL-LDAS Interface Coding Specification*, distributed as part of lalwrapper package and LIGO-T010003-00.

[2] Kent Blackburn *et al*, *The wrapper API baseline requirements and implementation*, LIGO-T990097-13.

[3] Bruce Allen *et al*, *Numerical Algorithms Library Specification and Style Guide*, distributed as part of lal package and LIGO-T990030.

Part II

LDAS-LAL coding specification and requirements

Chapter 2

LAL-LDAS Interface Coding Specification

2.1 Introduction

The LIGO Laboratory is building a modular data analysis pipeline called LDAS (LIGO Data Analysis System). Members of the LIGO Scientific Collaboration (and others) are writing LAL search code to analyze the data. This document describes the requirements on data analysis software intended for execution on the LDAS Beowulf clusters.

The interface between LDAS and LAL is a c++ executable called the `wrapperAPI`. In order to perform a particular search the `wrapperAPI` dynamically loads and calls four LAL functions that contain the search code. This document describes these LAL functions and lays out requirements on search code operating in the real-time LDAS analysis environment.

In brief summary, the logic of a search code must be encapsulated in these four functions which form a shared object library:

- `LALInitSearch()`
- `LALConditionData()`
- `LALApplySearch()`
- `LALFinalizeSearch()`

The `wrapperAPI` calls each of these functions in the order shown; it also has a built-in loop that repeatedly calls `LALApplySearch()` until the search is complete.

Since the `wrapperAPI` provides management and control structure for all parallel data analysis, the arguments of these functions are generic data type of sufficient flexibility to encompass the needs of all search codes. This document describes the arguments of these functions, i.e how LDAS will get the data to your analysis code and how your analysis code will pass the results back to LDAS.

2.1.1 The scope of this specification

This document formally defines the LAL functions that fit into the LDAS-LAL interface.[1] This is not a "how-to" manual for writing search code; however it is a "how it works document". Most importantly, this document spells out how information must flow from LDAS and `wrapperAPI` to the LAL search code.

[1] Some of the material in this document is redundant with the wrapperAPI baseline requirement document. If there is any discrepancy, that document takes precedence.

2.1.2 Applicability

The LAL Specification states "**all participating groups will be required to analyze LIGO data using LAL-compliant software.** However, at present, there is no similar requirement that states all groups will be required to analyze data in the LDAS environment. In spite of the lack of a formal rule, the LSC and LIGO Laboratory encourage developers to develop their code in such a way that it will fit into LDAS through the `wrapperAPI`. It must be emphasized that the `wrapperAPI` design does not impose significant constraints on algorithm design, although the need for reliable real-time analysis does impose certain reporting requirements.

2.1.3 The underlying design criteria for the interface

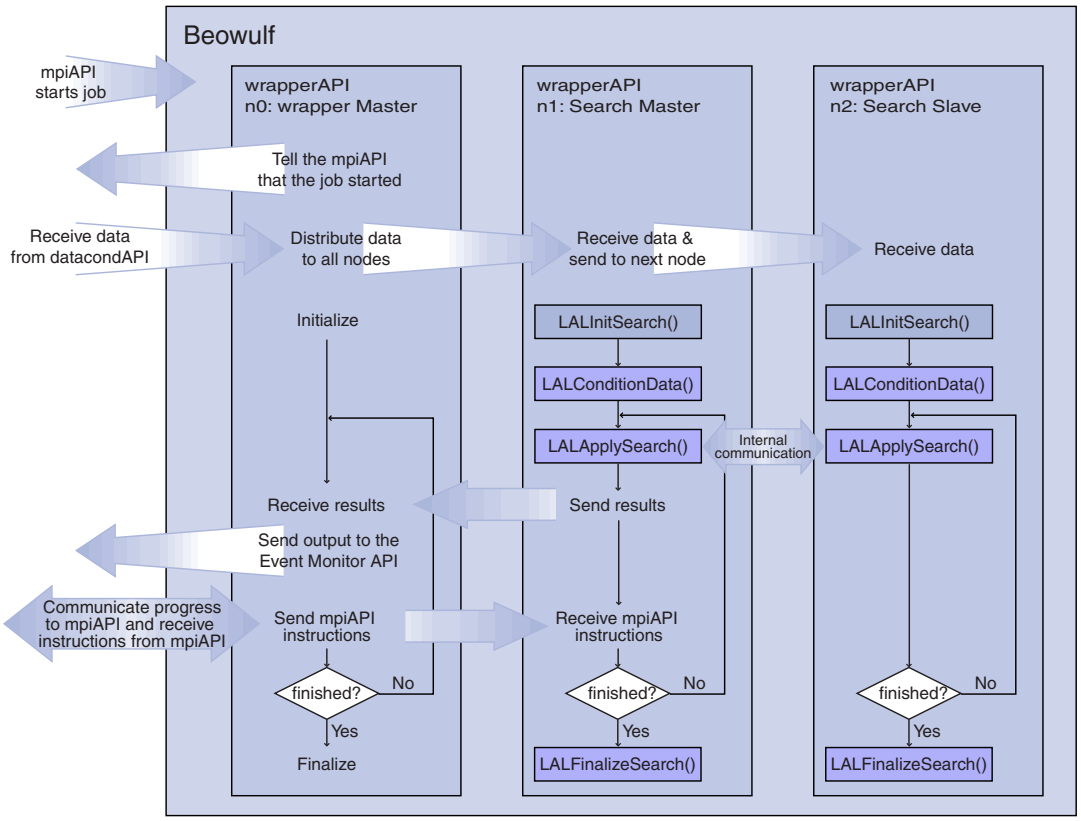
The requirements presented here strike a delicate balance between two competing design criteria. On the one hand, one wishes to keep the LDAS-LAL interface simple and flexibl. A simple, non-restrictive interface does not limit the types of algorithms that can be used within LDAS and will not limit the number of developers willing to climb the learning curve to fit there code into the slot. On the other hand, LDAS is designed to run multiple searches simultaneously, often analyzing data in real-time as it is acquired. This computing environment requires a strict rules for the search-code writers. The design of the `wrapperAPI` is a compromise between these two competing criteria.

The current interface gives developers complete control the computational and algorithmic aspects of their data analysis codes; the bulk of their code would be the same if they were writing standalone code. The two important differences between writing code for the LDAS environment and writing stand-alone code are:

- Developers are not responsible for reading data from frames within LDAS. Data flow is handled by LDAS and controlled by the developer via a user command and TCL script to execute it. This control layer is beyond the scope of this document.
- Developers are responsible for providing status and progress of the analysis at run-time – the mechanism is described below. In concert with the reporting requirements, code must support some management initiated and executed by the `wrapperAPI`.

Since the LDAS-LAL interface is flexible, it is hoped that data analysis code writers will install LDAS at their home institutions and do their code development within the LDAS environment. At the very least, the initial development of code to run under the `wrapperAPI` should be done using a standalone version installed on your local machine.

2.2 How does LAL fit into LDAS



2.3 Definition of the interface: the functions and the argument datatypes

In this section we explain how to the data and parameters get into your search code, and how you get the results out. Developers should be aware of one over-arching rule for their code: *All four functions must obey LAL coding standards as indicated in the LAL software specification.* The functions in each analysis code are LAL function, therefore, to the extent possible, they should adhere to the rules stated in Section 5 of the LAL Spec. Deviations from these rules are indicated below.

The interface to all search codes is the same by design. For this reason, most of the arguments are generic and do not change datatype from search to search. The exception is an argument of type `void`[2] in each of the functions. The developer will define a structure at the end of this pointer in the search header file. This is the structure used to port information between the four search functions. In the discussion below, the variable name assigned to the search-specific structure is `searchParams` and the data structure is (which the user must define) will be called `MySearchStruct`.

[2] In `LALApplySearch()` the `void *` is hidden inside the struct `LALApplySearchParams.`

2.3.1 Function LALInitSearch()

This functions is initiated on all nodes except the wrapperMaster. It is intended to parse the command line parameters (char *argv[]), and prepare them to be passed to the other functions. It should also be used to allocate memory for the search-specific data structures.

Prototype

```
void
LALInitSearch(
   LALStatus           *status,
   void               **searchParams,
   LALInitSearchParams *initSearchParams
);
```

Description

LALStatus *status: This is the standard LAL status structure. It is rigorously defined in the LAL Specification. Since this is a pure LAL function, the first argument is LALStatus.

void **searchParams: This void ** is a handle for your search-specific data structure, say MySearchStruct. Its purpose is to carry (pointers to) search-specific information (e.g. the data and the command line arguments) to your other functions. You must typedef this structure in your InitSearch.h header file, and attach it to the void ** in LALInitSearch(), e.g. *searchParams = LALMalloc(sizeof(MySearchStruct));

LALInitSearchParams *initSearchParams: The structure LALInitSearchParams is typedefed to the InitParams in LALWrapperInterface.h. The values in this data structure are prepared by the wrapperAPI and passed to LALInitSearch(); this structure definition cannot be changed in contributed analysis code. The elements of InitParams are:

initSearchParams -> argc is the argument count

initSearchParams -> argv[] is the character string containing the command line filter parameters. A valid parameter string must begin with initSearchParams->argv[0]="-filterparams" and will be followed by the search specific parameters, e.g. the string might look like -filterparams (1,2,3,(4,5),6.7,8.9). The wrapperAPI baseline requirements document give requirements for sanity checks of the parameters.

initSearchParams -> startTime is the start time of the wrapperAPI job in seconds since 0h 1 January 1970.

initSearchParams -> dataDuration is the total amount of data requested for this job rounded up to the nearest second.

initSearchParams -> realtimeRatio multiplied by dataDuration gives the maximum time alloted to this wrapperAPI job. If the analysis is not completed in this time, the job may be terminated by the mpiAPI.

initSearchParams -> rank indicates the node on which LALInitSearch was called. A value -1 indicates the wrapperMaster, 0 indicates a searchSlave, and 1 indicates the searchMaster.

In order to make use of the filterParameters information, LALInitSearch() parses argv and passes the information to the other functions using the "search specific" output structure that is attached to the void **.

Uses

Notes

2.3.2 Function LALConditionData()

This function is initiated on all nodes except the wrapperMaster. It is intended to perform search-specific data conditioning. If the run-time wrapperAPI option is set to -dataDistributor=C, then LALConditionData() will take responsibility for distribution of the data to the slaves with MPI communication. See note below. In general, LALConditionData() will unpack the input data from the generic *inout structure and insert it into the appropriate LAL data types.

Prototype

```
void
LALConditionData(
    LALStatus            *status,
    LALSearchInput       *inout,
    void                 *searchParams,
    LALMPIParams         *mpiParams
);
```

Description

LALStatus *status: This is the standard LAL status structure. It is rigorously defined in the LAL Specification. Since this is a pure LAL function, the first argument is LALStatus.

LALSearchInput *inout: This structure will contain your input data. It is typedefed in LALWrapperInterface.h to be type inPut which is defined in wrapperInterfaceDatatypes.h. The elements of inPut are:

inout -> numberSequences is the number of input sequences requested by the search code and packed into the multiDimData structure.

inout -> stateVector* TBD.

inout -> multiDimData* contains the input data. It is rigorously defined in the "Wrapper API's Baseline Requirements" http://www.ligo.caltech.edu/docs/T/T990097-15.pdf. You will need to unpack the multiDimData and repack it using LAL structures.

void *searchParams: This void * is a pointer to your search-specific data structure which you typedefed in InitSearch.h. Remember to recast it before you try to address its elements, i.e, localParams = (MySearchStruct *) searchParams ; This structure will contain the command line arguments that you parsed and attached in LALInitSearch. You should also attach the data you unpack from the multiDimData, as this structure will also be passed to LALApplySearch().

LALMPIParams *mpiParams: This is a pointer to the MPI parameters including the MPI communicator, which can be used to do MPI communication within the LALConditionData() routine. *Use of this argument is strongly discouraged: all MPI communication should be restricted to the LALApplySearch() function.* The MPI communicator can also be used to determine the number of nodes available using the MPI_Comm_size() function. Doing this is also discouraged as the result will be incorrect if the number of nodes available changes during execution. It is best to ignore this field altogether and do everything involving the MPI communicator in the LALApplySearch() function.

Uses

Notes

2.3.3 Function LALApplySearch()

This function is initiated on all nodes except the wrapperMaster. It should contain the computational engine of the analysis code. It is executed repeatedly (i.e inside a loop) by the wrapperAPI. On the searchMaster, it should return often to provide progress to the wrapperAPI and receive instructions from the mpiAPI. See Section 2.5.1 for a complete set of requirements.

Prototype

```
void
LALApplySearch(
   LALStatus            *status,
   LALSearchOutput      *output,
   LALSearchInput       *input,
   LALApplySearchParams *params
);
```

Description

LALStatus *status: This is the standard LAL status structure. It is rigorously defined in the LAL Specification. Since this is a pure LAL function, the first argument is LALStatus.

LALSearchOutput *output: This is the structure where you put your results. It is typedefed in LALWrapperInterface.h to be type SearchOutput which is defined in wrapperInterfaceDatatypes.h. The elements of SearchOutPut are:

```
typedef struct
{
  INT4    numOutput;
  outPut* result;
  REAL4   fracRemaining;
  BOOLEAN notFinished;
}SearchOutput;
```

The elements of SearchOutput are:

output -> numOutput is the number of events that are packed into result.

output -> result is the structure into which you pack your event information. The type outPut is defined in the "Wrapper API's Baseline Requirements" http://www.ligo.caltech.edu/docs/T/T990097-15.pdf.

output -> fracRemaining fraction of analysis remaining to be done when LALAppySearch returns; remember, this function is called many times. Developers should insure that they have a routine which computes this number as accurately as possible. This need only be set on the searchMaster.

output -> notfinished indicates that the analysis is complete if set to FALSE, i.e. LALAppySearch should not be called again on the node which returned this value. Otherwise it should be set to TRUE on any node that needs LALAppySearch called again.

LALSearchInput *input: Same as LALSearchInput structure described above. For some simple searches, there may be no need for data conditioning, and therefore the unpacking of the input data can be postponed until LALApplySearch() is called.

LALApplySearchParams *params: This contains the search specific information. It is is defined in LALWrapperInterface.h

```
typedef struct
tagLALApplySearchParams
{
  LALMPIParams *mpiParams;
```

```
  void         *searchParams;
}
LALApplySearchParams;
```

The elements of LALApplySearchParams are:

params -> mpiParams provide the MPI_Comm and some wrapperAPI-centric information. LALMPIParams is typedefed to SearchParams in LALWrapperInterface.h; in wrapperInterfaceDatatypes.h SearchParams is typedefed to be

```
typedef struct
{
  MPI_Comm*     comm;   /* wrapper slave COMM_WORLD */
  MPIapiAction* action; /* instruction from mpiAPI to search code */
}SearchParams;
```

params -> mpiParams -> comm the MPI communicator containing information about the searchMaster and searchSlave nodes.

params -> mpiParams -> action is needed when the mpiAPI will execute dynamic load-balancing. It is typedefed to be

```
typedef struct
{
  INT4    add;       /* number of nodes added or subtracted */
  BOOLEAN mpiAPIio;  /* command from mpiAPI: false - exit, true - continue */
}MPIapiAction;
```

where add is the number of nodes added to, or subtracted from, the communicator when load-balancing was executed. The BOOLEAN mpiAPIio is a flag which indicates if the node will continue, or that the code has been instructed to exit by the mpiAPI.

params -> searchParams is the void * pointer to the same search-specific structure discussed for the previous functions. As before, it should be recast, i.e, localParams = (MySearchStruct *) searchParams ;.

Uses

Notes

2.3.4 Function LALFinalizeSearch()

Frees the memory allocated for the search-specific datatype **searchParams.

Prototype

```
void
LALFinalizeSearch(
   LALStatus           *status,
   void               **searchParams
);
```

Description

LALStatus *status: This is the standard LAL status structure. It is rigorously defined in the LAL Specification. Since this is a pure LAL function, the first argument is LALStatus.

void **searchParams: This void ** is a handle for your search-specific data structure, say MySearchStruct.

Notes

Need to briefly explain the two modes of data distribution.

2.4 LALWrapper code Organization

This section explains the layout of code within the LALWrapper. First we give the large-scale structure: the directory tree. Then we describe the finer structure: the required format and content of the individual source files. The code and the documentation are inextricably entwined: the hierarchy of the code elements (packages, headers, modules) determines the hierarchy of the documentation (chapters, sections, subsections). Even at finer resolution this holds: the contents of the individual source files also matches the content of the individual documentation pieces.

Note that the source code that makes up the shared object Libraries are genuine LAL functions, and therefore the requirements for code organization are the those given in the Chapter 6 of LALspec. There is currently one exception: a `/test` directory is not required in LALWrapper since that would require a standalone wrapperAPI to be installed locally. (This rule may change as the code matures.) In the LAL distribution this directory contains the executables that drive the routines for testing purposes.

2.4.1 Organization of contributed analysis codes

All LALWrapper components will reside in a single directory, called `lalwrapper`, and its subdirectories. The LSC Software Coordinator and Software Librarian will maintain an official master copy of the LALWrapper source in the CVS repository. An official release will be distributed as a tar-ball thru the LAL web pages. Users can download and install a release on their own machines. Within this top level directory, their will be a subdirectory (`lalwrapper/contrib/`) where the analysis codes will reside. Within this subdirectory, every analysis code will have a named directory that contains all files associated with that analysis code (e.g. lalwrapper/contrib/inspiral). The development of contributed analysis code will be the primary way collaborators will contribute to the LALWrapper. A contributed analysis code (e.g. lalwrapper/contrib/mysearch) should have the source files, documentation and Makefiles in the following subdirectories:

- `lalwrapper/contrib/mysearch/include`: All the header files associated with `mysearch`. The header file `MySearch.h` should define the `MySearchStruct` type which is needed by the four search functions. Each of the four required functions should have associated header files (e.g. `MySearchInitSearch.h`, `MySearchConditionData.h`, etc). Other header files can be present at the discretion of the developer, however, all header files must conform to the format and style described in Section 6.2.1 of the LAL specification.

- `lalwrapper/contrib/mysearch/src`: all the source files associated with the component. Each of the four required functions should have associated source files (e.g. `MySearchInitSearch.c`, `MySearchConditionData.c`, etc). Other source files can be present at the discretion of the developer, however, all source files must conform to the format and style described in Section 6.2.2 of the LAL specification. In particular, each source file must be associated with one (and only one) header file.

- `lalwrapper/contrib/mysearch/test`: test scripts and all supporting files associated with component-level tests. The format and style of these tests remains TBD as LAL, LALWrapper and LDAS mature.

- `lalwrapper/contrib/mysearch/doc`: There will be a LaTeX file in this directory capable of assembling stand-alone documentation for this package. There will also be a LaTeX file that forms a chapter in comprehensive manual for the entire LALWrapper. Before auto-extraction with `laldoc`, much of the text source for the documentation may reside in the code files. See Section 7 of the LAL specification for details.

2.5 The rules for contributed analysis code in LALWrapper

Contributed analysis codes must meet certain standards before they will be admitted to the LDAS real-time computing environment. These rules must be met by all analysis codes in LALWrapper

in production releases. As LALWrapper evolves from development to production software, more of these rules will be gradually enforced. In this section, we indicate which rules must be obeyed by contributed code at this time. If the upcoming release is greater than or equal to the release number listed with each requirement, then analysis code must adhere to that rule.

2.5.1 The Rules (and the reasons for the rules) for Real-Time Computing in the LDAS

When running analysis along side other analyses in real-time it is not enough to simply conform to the interface specification in the previous sections. The real-time LDAS computing environment is high-speed, communal computing in close quarters: there must be strict rules. In this section, we present a set of requirements and restrictions for search codes running within this environment.

1. Each type of analysis must be executed as a sequence of short jobs which can be completed in a short time, i.e 1–few hours. For example, a search may bite off an hours worth of data, analyze it, terminate, and then restart with the next hours worth of data.
   - Reason 1: To avoid job failure due to secular memory leaks.
   - Reason 2. Load balancing. Although there will be means to reassign nodes to running jobs, short execution times allow the possibility to redistribute resources in a timely manner.
2. Progress through the analysis must be periodically reported to the `wrapperAPI`. To achieve this, the search function `LALApplySearch()` is called by the `wrapperAPI`in a while loop that ends when `notFinished` is set to `FALSE`. `LALApplySearch()` should return at least 10 times per job (or every $\sim$ 5 minutes on all nodes. value between zero and unity should be assigned to `fracRemaining` on the `wrapperMaster` each time `LALApplySearch()` returns. fraction remaining.
   - Reason 1: The wrapper may need to make load balancing decisions while the job is still running.
   - Reason 2: If a job hangs, and doesn't report for a long time the LDAS job management system needs to be able to figure it out and kill the job.

2.6 LALWrapper code documentation

All the functions written by developers should be LAL functions, and therefore the philosophy, the requirements and the organization of the documentation are those given in the Chapter 7 of LAL Specification. In particular, the same auto-documentation tool, `laldoc` will be used.
In summary, the documentation requirements are:

- Documentation will be written in LaTeX.
- The author and CVS Id block in the code must be auto-extracted from the code and automatically included in the documentation.
- Error codes and error descriptions must be auto-extracted from the header files and automatically included in the documentation.
- Function prototypes must be auto-extracted and included in the documentation.
- All functions must be entered into the LaTeX document index with an \index{} command.
- All non-LAL data structures must be entered into the LaTeX document index with an \index{} command.
- Do not let the code get lost in the documentation.

2.7 Maintaining the LALWrapper

2.7.1 Version control for LALWrapper

The LL and LSC will jointly maintain both the LALWrapper software and this specification. The source code and documentation – and this document – will be kept in a CVS repository. When a package is submitted to the library its directory tree will be entered in the CVS repository. The revision history of the files will be available on the web. The LSC Software Coordinator and Software Librarian will over see the day-to-day operations of the repository. They will also see that the most up-to-date versions of all code files are publicly available on the web.

2.7.2 Numbering the LALWrapper releases

The numbering of versions and releases of the library will done according to the same rules as the LAL specification.

2.7.3 Validation of LALWrapper code

Verifying that the individual components (functions) work will primarily be the responsibility of the code developers. The LSC Software Coordinator, the LSC data analysis subgroup chairs and the LL personnel will organize integrated tests of the analysis pipeline through mock data challenges. These tests will be conducted to validate the code in LALWrapper. A complete record of these challenges will be maintained in a CVS repository which collects together a final report, test checklists completed during the MDC, scripts and test programs, input data, output data and correct results. Each MDC should result in a set of tests which can be used to validate later releases of the analysis codes.

2.7.4 Requesting changes in LALWrapper

The LIGO Laboratory and LSC will maintain a web page for submitting bug reports and releasing the code. Currently, this can be found at http://www.lsc-group.phys.uwm.edu/lal.

While in the development phase, updates to code and documentation will be the responsibility of the developers. However, as we transition to production releases, the procedure for updating code will become more formal. [During the early stages a-c will apply. In the more formal stage a-e apply.]

1. All modified code will be verified (and validated in a pipeline test if necessary). All affected documentation will be revised to show changes.
2. Once available, a new release will be distributed.
3. A history of revisions shall be maintained and made available to users.
4. Change requests will be reviewed jointly by LL and LSC on a regular basis.
5. Those changes which are selected for incorporation shall be assigned for implementation to respective groups.

2.8 Development tools and software packages required for shared object development

As the LALWrapper shared object library is a library of LAL functions, the development tools should be the same as those required for LAL, e.g. GNU CVS, Linux [Redhat 6.0 or greater], gcc, etc. In addition, some parts of the LDAS may be required.

Part III

The LAL-Wrapper Interface

Chapter 3

Basics of wrapperAPI

Chapter 4

The interface between LAL and wrapperAPI

4.1 Header LALWrapperInterface.h

Provides the generic LAL function interface with wrapperAPI.

Synopsis

```
#include <LALWrapperInterface.h>
```

This header covers the generic LAL routines that are ultimately called by wrapperAPI. These routines have the same form for all searches, though their implementation is search specific.

Structures

```
struct LALSearchInput
struct LALSearchOutput
struct LALInitSearchParams
struct LALMPIParams
struct LALApplySearchParams
```

Prototypes

1.80
LALWrapperInterface.h

```
void
LALInitSearch(
    LALStatus             *status,
    void                 **searchParams,
    LALInitSearchParams   *initSearchParams
    );


void
LALConditionData(
    LALStatus             *status,
    LALSearchInput        *inout,
    void                  *searchParams,
    LALMPIParams          *mpiParams
    );


void
LALApplySearch(
    LALStatus             *status,
    LALSearchOutput       *output,
    LALSearchInput        *input,
    LALApplySearchParams  *params
    );


void
LALFinalizeSearch(
    LALStatus             *status,
    void                 **searchParams
    );
```

These are the generic LAL routines that form the LAL-Wrapper interface.

LALInitSearch

LALConditionData

LALApplySearch

LALFinalizeSearch

4.1.1 Module LALWrapperInterface.c

This module provides the routines called by wrapperAPI which "wrap" the generic LAL routines. This module, therefore, is not search specific.

The general tasks that this module perfoms are:

- Defines lalDebugLevel.
- Defines a LALStatus variable for each of the LAL-Wrapper interface generic functions.
- Defines a global variable that contains specific search parameters, which is created by LALInitSearch() and destroyed by LALFinalizeSearch(), and is passed to LALApplySearch() and LALConditionData().
- Re-packages variables given by wrapperAPI into the structures used by the LAL-Wrapper interface generic functions.
- Parses the exit status from the LAL-Wrapper interface generic functions into a character string used by wrapperAPI.
- Provides a routine for freeing output memory declared within the search specific LAL routines.
- Handles various signals that may be raised by LAL.

4.1.2 Program happyAPI.c

A program that does a crude approximation to what wrapperAPI does: dynamically loads a shared-object library and executes the functions required by wrapperAPI. This program is used to test a shared object that is to be run by wrapperAPI.

Usage

```
happyAPI sofile [filterparams]
```

where sofile is the shared object library (lib⟨whatever⟩.so) to test and filterparams are the optional filter parameters.

Description

Exit codes

Code	Explanation
0	Success, normal exit.
1	An error was returned by a shared object function.
2	An internal error.

4.2 Header ExtractSeries.h

Provides LAL functions for extracting time and frequency series from a LALSearchInput structure.

Synopsis

```
#include <ExtractSeries.h>
```

This header covers LAL routines that are used to extract time and frequency series from the multiDimData sequences in a LALSearchInput structure. These routines are typically called in LALConditionData().

Error conditions

<*name*>	code	description
NULL	001	"Null pointer"
NNUL	002	"Non-null pointer"
ZSEQ	004	"Zero sequences"
MTCH	010	"No matching sequence found"
TYPE	020	"Wrong data type"
SERT	040	"Wrong series type"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in ExtractSeries.h on line 1.43.

4.2.1 Module ExtractSeries.c

This module provides the routines used to extract time and frequency series from the list of sequences of multiDimData contained in a LALSearchInput structure. Sequences are selected via a regular expression pattern which is used to examine the names of the sequences. The sequence that first matches the pattern is then extracted.

Prototypes

1.84 ExtractSeries.c
```
void
LALFindSequenceByName(
    LALStatus       *status,
    multiDimData   **output,
    LALSearchInput  *input,
    const CHAR      *regex
    )
```

1.127 ExtractSeries.c
```
void
LALFindSequenceByComment(
    LALStatus       *status,
    multiDimData   **output,
    LALSearchInput  *input,
    const CHAR      *regex
    )
```

1.225 ExtractSeries.c
```
void
LALExtractINT2TimeSeries (
    LALStatus           *status,
    INT2TimeSeries       *series,
    multiDimData        *input
    )
```

1.299 ExtractSeries.c
```
void
LALExtractINT2FrequencySeries (
    LALStatus                   *status,
    INT2FrequencySeries        *series,
    multiDimData                *input
    )
```

1.375 ExtractSeries.c
```
void
LALExtractINT4TimeSeries (
    LALStatus           *status,
    INT4TimeSeries       *series,
    multiDimData        *input
    )
```

1.449 ExtractSeries.c
```
void
LALExtractINT4FrequencySeries (
    LALStatus                   *status,
    INT4FrequencySeries        *series,
    multiDimData                *input
    )
```

1.525 ExtractSeries.c

```
void
LALExtractINT8TimeSeries (
    LALStatus           *status,
    INT8TimeSeries       *series,
    multiDimData        *input
    )
```

1.599 ExtractSeries.c

```
void
LALExtractINT8FrequencySeries (
    LALStatus                   *status,
    INT8FrequencySeries        *series,
    multiDimData                *input
    )
```

1.675 ExtractSeries.c

```
void
LALExtractUINT2TimeSeries (
    LALStatus           *status,
    UINT2TimeSeries       *series,
    multiDimData        *input
    )
```

1.749 ExtractSeries.c

```
void
LALExtractUINT2FrequencySeries (
    LALStatus                   *status,
    UINT2FrequencySeries        *series,
    multiDimData                *input
    )
```

1.825 ExtractSeries.c

```
void
LALExtractUINT4TimeSeries (
    LALStatus           *status,
    UINT4TimeSeries       *series,
    multiDimData        *input
    )
```

1.899 ExtractSeries.c

```
void
LALExtractUINT4FrequencySeries (
    LALStatus                   *status,
    UINT4FrequencySeries        *series,
    multiDimData                *input
    )
```

1.975 ExtractSeries.c

```
void
LALExtractUINT8TimeSeries (
    LALStatus           *status,
    UINT8TimeSeries       *series,
    multiDimData        *input
    )
```

1.1049 ExtractSeries.c

```
void
LALExtractUINT8FrequencySeries (
    LALStatus                   *status,
    UINT8FrequencySeries        *series,
    multiDimData                *input
    )
```

1.1125 ExtractSeries.c

```
void
LALExtractREAL4TimeSeries (
    LALStatus           *status,
    REAL4TimeSeries       *series,
    multiDimData        *input
    )
```

1.1199 ExtractSeries.c

```
void
LALExtractREAL4FrequencySeries (
    LALStatus                   *status,
    REAL4FrequencySeries        *series,
    multiDimData                *input
    )
```

1.1275 ExtractSeries.c

```
void
LALExtractREAL8TimeSeries (
    LALStatus           *status,
    REAL8TimeSeries       *series,
    multiDimData        *input
    )
```

1.1349 ExtractSeries.c

```
void
LALExtractREAL8FrequencySeries (
    LALStatus                   *status,
    REAL8FrequencySeries        *series,
    multiDimData                *input
    )
```

1.1425 ExtractSeries.c

```
void
LALExtractCOMPLEX8TimeSeries (
    LALStatus           *status,
    COMPLEX8TimeSeries       *series,
    multiDimData        *input
    )
```

1.1499 ExtractSeries.c

```
void
LALExtractCOMPLEX8FrequencySeries (
    LALStatus                   *status,
    COMPLEX8FrequencySeries        *series,
    multiDimData                *input
    )
```

1.1575 ExtractSeries.c

```
void
LALExtractCOMPLEX16TimeSeries (
    LALStatus           *status,
    COMPLEX16TimeSeries       *series,
    multiDimData        *input
    )
```

1.1649 ExtractSeries.c

```
void
LALExtractCOMPLEX16FrequencySeries (
    LALStatus                   *status,
    COMPLEX16FrequencySeries        *series,
    multiDimData                *input
    )
```

The routines LALFindSequenceByName() and LALFindSequenceByComment() examine the input LALSearchInput structure by looping over the input multiDimData sequences. For each sequence, the routine examines the name or comment (depending on which of the two functions is called) of the routine to see if it matches the specified regular expression pattern. The first sequence that so matches is then extracted and a pointer to this sequence is returned.

The routines LALExtract⟨*type*⟩TimeSeries() and LALExtract⟨*type*⟩FrequencySeries() convert an input multiDimData structure into a LAL time or frequency series. All of the appropriate metadata contained in the multiDimData sequence is copied into the output time or frequency series. This includes the name, the start epoch, the base frequency, the time or frequency step size, and the number of points of data. However, the units are not presently copied. They must be set manually by the user (either before or after the routine is called).

Depending on the initial condition of the output structure, the code will behave in different ways. If the time or frequency series data field (which is a LAL vector) is NULL then the memory for the vector is allocated using the appropriate LAL creation routine. In this case, the data is then *copied* into the newly allocated memory. This memory must be freed when the user is finished with it (using the appropriate LAL destroy vector function).

However, if the time or frequency series data field is not NULL, but the data field of this vector *is* NULL, then the routine simply attaches the data pointer from the multiDimData structure to the vector's data field. Thus, there is no additional memory allocation. However, this data should be treated as readonly! The data contained in the input (and thus in the time or frequency series) should not be modified. Furthermore, this memory should not be freed. This will be done automatically by the wrapperAPI. Using this method of accessing data is somewhat advantageous in terms of memory use, but this method should only be used if the data will processed non-destructively into another form in new memory allocated to hold it. E.g., if the input time series data is immediately (and out-of-place) FFTed and the results put into a frequency series with memory allocated to hold it, and then the time series is never used again, then this method is suitable. If this is not clear, use the former method.

If neither the data field of the time or frequency series is NULL nor the data field of the data field of the time or frequency series (i.e., the data field of the LAL vector, which is the data field of the time or frequency series) is NULL, then the routine will return an error.

4.2.2 Module Calibration.c

This module contains code used to extract calibration information contained in a LALSearchInput structure, and to construct a response (or transfer) function. This is supposed to provide a high-level interface for DSO authors to obtain a response function in the desired form.

Prototypes

1.60 Calibration.c

```
void
LALExtractResponse(
    LALStatus               *status,
    COMPLEX8FrequencySeries *output,
    LALSearchInput          *input,
    const CHAR              *ifo
    )
```

The routine LALExtractResponse() extracts the necessary calibration information from the LALSearchInput structure and constructs a response function (as a frequency series) from this information. If the fourth argument is non-NULL then this string specifies the detector (H1, H2, L1, etc.) for which the calibration is required; if this argument is NULL, the first calibration information found will in the LALSearchInput will be used.

Certain fields of the output should be set before this routine is called. In particular: 1. The epoch field of the frequency series should be set to the correct epoch so that the routine can generate a response function tailored to that time (accounting for calibration drifts). 2. The units of the response function should be set to be either strain-per-count (for a response function) or count-per-strain (for a transfer function); the routine will then return either the response function or its inverse depending on the specified units. Furthermore, the power-of-ten field of the units is examined to scale the response function accordingly. 3. The data vector should be allocated to the required length and the frequency step size should be set to the required value so that the routine can interpolate the response function to the required frequencies.

Note that the LALSearchInput must contain the appropriate calibration multiDimData sequences, which must be input to the wrapperAPI.

4.3 Header BuildDB.h

Provides LAL functions for building database entries.

Synopsis

```
#include <BuildDB.h>
```

This header covers LAL routines that are used to build dataBase entries to be returned to the wrapperAPI by LALApplySearch().

Error conditions

<*name*>	code	description
NULL	001	"Null pointer"
NNUL	002	"Non-null pointer"
SIZE	004	"Invalid data length"
ALOC	010	"Memory allocation error"
ROWS	020	"Invalid number of rows"
TYPE	040	"Invalid datatype"

The status codes in the table above are stored in the constants BUILDDBH_E<*name*>, and the status descriptions in BUILDDBH_MSGE<*name*>. The source code with these messages is in BuildDB.h on line 1.42.

Structures

Type LALBLOB

1.51 BuildDB.h

```
typedef struct
tagLALBLOB
{
  UINT4  size;
  void  *data;
}
LALBLOB;
```

LAL container for a Binary Large OBject (BLOB). The fields are

`size` The size, in bytes, of the BLOB.

`data` Pointer to the memory containing the BLOB.

Type SearchSummaryInfo

1.71 BuildDB.h

```
typedef struct
tagSearchSummaryInfo
{
  const char *dsoName;
  const char *comment;
  LIGOTimeGPS inputStartTime;
  LIGOTimeGPS inputEndTime;
  LIGOTimeGPS outputStartTime;
  LIGOTimeGPS outputEndTime;
  UINT4       numEvents;
  UINT4       numNodes;
}
SearchSummaryInfo;
```

Summary information about a search required by the function LALCreateSearchSummaryDataBase() to populate the search_summary database table. The fields are:

dsoName The name of the DSO (no more than 64 characters).

comment User comment (no more than 240 characters).

inputStartTime The GPS start time of the input data.

inputEndTime The GPS end time of the input data.

outputStartTime The GPS start time of the data actually analyzed.

outputEndTime The GPS end time of the data actually analyzed.

numEvents The number of events found.

numNodes The number of nodes this job used.

Type SearchSummaryVariables

1.106
BuildDB.h

```
typedef struct
tagSearchSummaryVariables
{
  const char  *name;
  const char  *string;
#ifdef BUILDBH_LDAS_0_6
  const REAL8 *value;
#else
  const REAL4 *value;
#endif
}
SearchSummaryVariables;
```

Summary variables generated by a search required by the function LALCreateSearchSummvarsDataBase() to populate the search_summvars database table. The fields are:

name The name of the variable.

string The string associated with the variable (NULL to omit)

value The floating point value of the variable (NULL to omit)

4.3.1 Module BuildDB.c

This module provides the routines used to create dataBase entries required as output from LALApplySearch().

Prototypes

```
void
LALCreateBLOB(
    LALStatus   *status,
    LALBLOB    **output,
    const void  *input,
    UINT4        size
    )
```
1.290 BuildDB.c

```
void
LALDestroyBLOB(
    LALStatus  *status,
    LALBLOB   **blob
    )
```
1.327 BuildDB.c

```
void
LALCreateNextDBEntry(
    LALStatus   *status,
    dataBase   **output,
    const CHAR  *table,
    const CHAR  *column
    )
```
1.344 BuildDB.c

```
void
LALCreateDBEntryData(
    LALStatus *status,
    dataBase  *output,
    datatype   type,
    UINT4      rows
    )
```
1.369 BuildDB.c

```
void
LALCreateDBEntryBLOB(
    LALStatus *status,
    dataBase  *output,
    LALBLOB   *blob,
    UINT4      rows
    )
```
1.395 BuildDB.c

```
void
LALCreateDBEntryUBLOB(
    LALStatus *status,
    dataBase  *output,
    LALBLOB   *blob,
    UINT4      rows
    )
```
1.434 BuildDB.c

1.473 BuildDB.c

```
void
LALCreateSearchSummaryDataBase(
    LALStatus          *status,
    dataBase          **output,
    SearchSummaryInfo  *info
    )

void                                                          1.578
LALCreateSearchSummvarsDataBase(                              BuildDB.c
    LALStatus               *status,
    dataBase               **output,
    SearchSummaryVariables  *vars
    )
```

Description

The routine LALCreateBLOB() copies a Binary Large OBject (BLOB), given as input along with its size in bytes, to a LALBLOB structure, which is created. The routine LALDestroyBLOB() destroys a LALBLOB structure.

The routine LALCreateNextDBEntry() allocates memory for a new dataBase structure, a pointer to this structure is returned as the output. If the output is not a pointer to NULL, the new dataBase is attached to the existing linked list (the pointer to the next structure must be NULL). As input this routine requires the table name and the column name for the database entry.

The routine LALCreateDBEntryData() allocates memory for the data to be contained in the dataBase structure. This routine takes as input the type and number of rows of data that will be put into the database entry column. After the routine is called, the calling routine must populate the data array allocated by this routine. If a Binary Large OBject (BLOB) is to be entered into the database, use LALCreateDBEntryBLOB() instead. If your blob has binary data, not just ascii characters, you should use LALCreateDBEntryUBLOB() as this will represent the data in octal form rather than as characters (which may be unprintable). This routine creates a number of copies of the BLOB as there are rows in the column. (There is no routine to put different BLOBs in the different rows; this must be done by hand.)

The routine LALCreateSearchSummaryDataBase() is a higher-level routine that creates and populates a dataBase structure containing all the information required for a search_summary database entry.

The routine LALCreateSearchSummvarsDataBase() is a higher-level routine that creates and populates a dataBase structure containing all the information required for a search_summvars database entry.

There are no routines to destroy the dataBase structures created by these routines. This is done automatically by the LALWrapperInterface after the wrapperAPI has processed them.

Example

Building output in LALApplySearch() can be somewhat cumbersome. Here one example of a output building routine. It creates two output results, one is the search_summary database entry, the second is a sgnl_unmodeled database entry. The latter consists of a number of columns, each with a number of rows equal to the number of events (each event is a row in the sgnl_unmodeled table).

```
typedef struct tagSillyEvent
{
  LIGOTimeGPS  epoch;
  REAL4        duration;
  REAL4        amplitude;
  REAL4        snr;
```

```
  REAL4        confidence;
}
SillyEvent;

typedef struct tagSillyResults
{
  CHAR         ifoName[3];
  CHAR         channel[64];
  LIGOTimeGPS  inputStartTime;
  LIGOTimeGPS  inputEndTime;
  LIGOTimeGPS  outputStartTime;
  LIGOTimeGPS  outputEndTime;
  UINT4        numNodes;
  UINT4        numEvents;
  SillyEvent  *events;
}
SillyResults;

void
SillyBuildOutput(
    LALStatus       *status,
    LALSearchOutput *output,
    SillyResults    *input
    )
{
  CHAR                    searchName[] = "silly";
  CHAR                    varName[] = "silliness";
  LALBLOB                 blob;
  SearchSummaryInfo       info;
  SearchSummaryVariables  summvars;
  dataBase               *dbase;
  UINT4                   event;

  INITSTATUS( status, "SillyBuildOutput", "Silly: $Id: BuildDB.c,v 1.8 2003/01/20 22:50:53 whelan Exp $" );
  ATTATCHSTATUSPTR( status );

  output->numOutput = 3;
  output->result = LALCalloc( output->numOutput, sizeof( *output->result ) );
  if ( ! output->result )
  {
    ABORT( status, 1, "Allocation error" );
  }
  output->result[0].search = burst;
  output->result[1].search = burst;
  output->result[2].search = burst;
  output->result[0].significant = 1;
  output->result[1].significant = 1;
  output->result[2].significant = 1;

  /* first output: the search_summary database entry */
  dbase = NULL;
  info.dsoName         = searchName;
  info.inputStartTime  = input->inputStartTime;
  info.inputEndTime    = input->inputEndTime;
```

```
info.outputStartTime = input->outputStartTime;
info.outputEndTime   = input->outputEndTime;
info.numEvents       = input->numEvents;
info.numNodes        = input->numNodes;
LALCreateSearchSummaryDataBase( status->statusPtr, &dbase, &info );
CHECKSTATUSPTR( status );
output->result[0].results = dbase;

/* second output: the search_summary database entry */
dbase = NULL;
summvars.name        = varName;
summvars.value       = silliness;
summvars.string      = NULL;
LALCreateSearchSummvarsDataBase( status->statusPtr, &dbase, &summvars );
CHECKSTATUSPTR( status );
output->result[1].results = dbase;

/* third output: the sgnl_unmodeled database entry with numEvent rows */
dbase = NULL;

/* column "search" */
LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "search" );
CHECKSTATUSPTR( status );
output->result[2].results = dbase; /* bind head of list to the output */
blob.data = search;
blob.size = strlen( search );
LALCreateDBEntryBLOB( status->statusPtr, dbase, &blob, input->numEvents );
CHECKSTATUSPTR( status );

/* column "ifo" */
LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "ifo" );
CHECKSTATUSPTR( status );
blob.data = input->ifoName;
blob.size = 2;
LALCreateDBEntryBLOB( status->statusPtr, dbase, &blob, input->numEvents );
CHECKSTATUSPTR( status );

/* column "channel" */
LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "channel" );
CHECKSTATUSPTR( status );
blob.data = input->channel;
blob.size = strlen( input->channel );
LALCreateDBEntryBLOB( status->statusPtr, dbase, &blob, input->numEvents );
CHECKSTATUSPTR( status );

/* column "start_time" */
LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "start_time" );
CHECKSTATUSPTR( status );
LALCreateDBEntryData( status->statusPtr, dbase, int_4s, input->numEvents );
CHECKSTATUSPTR( status );
for ( event = 0; event < input->numEvents; ++event )
{
  dbase->rows.int4s[event] = input->events[event].epoch.gpsSeconds;
}
```

```
  /* column "start_time_ns" */
  LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "start_time_ns" );
  CHECKSTATUSPTR( status );
  LALCreateDBEntryData( status->statusPtr, dbase, int_4s, input->numEvents );
  CHECKSTATUSPTR( status );
  for ( event = 0; event < input->numEvents; ++event )
  {
    dbase->rows.int4s[event] = input->events[event].epoch.gpsNanoSeconds;
  }

  /* column "duration" */
  LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "duration" );
  CHECKSTATUSPTR( status );
  LALCreateDBEntryData( status->statusPtr, dbase, real_4, input->numEvents );
  CHECKSTATUSPTR( status );
  for ( event = 0; event < input->numEvents; ++event )
  {
    dbase->rows.real4[event] = input->events[event].duration;
  }

  /* column "amplitude" */
  LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "amplitude" );
  CHECKSTATUSPTR( status );
  LALCreateDBEntryData( status->statusPtr, dbase, real_4, input->numEvents );
  CHECKSTATUSPTR( status );
  for ( event = 0; event < input->numEvents; ++event )
  {
    dbase->rows.real4[event] = input->events[event].amplitude;
  }

  /* column "snr" */
  LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "snr" );
  CHECKSTATUSPTR( status );
  LALCreateDBEntryData( status->statusPtr, dbase, real_4, input->numEvents );
  CHECKSTATUSPTR( status );
  for ( event = 0; event < input->numEvents; ++event )
  {
    dbase->rows.real4[event] = input->events[event].snr;
  }

  /* column "confidence" */
  LALCreateNextDBEntry( status->statusPtr, &dbase, "sngl_unmodeled", "confidence" );
  CHECKSTATUSPTR( status );
  LALCreateDBEntryData( status->statusPtr, dbase, real_4, input->numEvents );
  CHECKSTATUSPTR( status );
  for ( event = 0; event < input->numEvents; ++event )
  {
    dbase->rows.real4[event] = input->events[event].snr;
  }

  DETATCHSTATUSPTR( status );
  RETURN( status );
}
```

```
* $Id: BuildDB.c,v 1.8 2003/01/20 22:50:53 whelan Exp $
```

1.1
BuildDB.c

Part IV

Contributed Shared Objects

Chapter 5

Shared object exttrig

This is the introductory section. It describes the motivation for searching for bursts using the External Triggers.

5.1 Description

This section describes the method used for the External Trigger burst search.

5.2 Command Line Arguments

The command line arguments for the external trigger search code follow. The libldasexttrig.so module should be used with the following arguments to the wrapperAPI:

argv[0] = "-filterparms"

argv[1] = gpsSeconds The seconds field for the GPS start time for the external trigger event.

argv[2] = gpsNSeconds The nanoseconds field for the GPS start time for the external trigger event.

argv[3] = sourceRA The right ascension angle of the external trigger event source.

argv[4] = sourceDec The declination of the external trigger event source.

argv[5] = inputWindow Window/segment length to cross-correlate (in seconds).

argv[6] = inputOnShift The time interval between the end of the 'on'-source segment and the beginning of the external trigger event.

argv[7] = inputOffShiftLow The hard lower limit for the interval between the beginning of the external trigger event and the beginning of the 'off'-source segments.

argv[7] = inputOffShiftHigh The hard upper limit for the interval between the beginning of the external trigger event and the end of the 'off'-source segments.

argv[9] = dqFlag A flag whether (1) or not (0) to use the data quality information.

5.3 Header ExtTrig.h

Provides ETSearchParams structure to the various LALwrapperInterface standard functions.

Synopsis

```
#include "ExtTrig.h"
```

Error Conditions

There are no error conditions associated with this header file.

Structures

struct ETSearchParams

Common parameters for parallel processing

BOOLEAN searchMaster

INT4 rank

UINT4 numSlaves

UINT4 curSlaves

CHAR chan1[dbNameLimit]

CHAR chan2[dbNameLimit]

Containers for the search parameters (to be output to table)

LIGOTimeGPS grbStart The GPS start times of the GRB events

SkyPosition sourceDir The direction of the GRB event

REAL4 inputWindow The time series segment to cross-correlate (in seconds)

REAL4 inputOnShift The time interval between the end of an 'on'-source segment and the beginning of a GRB event

REAL4 inputOffShiftLow The hard lower limit for the interval between the beginning of a GRB event and the beginning of the 'off'-source segment

REAL4 inputOffShiftHigh The hard upper limit for the interval between the beginning of a GRB event and the end of an 'off'-source segment

BOOLEAN dqFlag A flag whether (1) or not (0) to use the data quality information obtained.

Containers for the datacondAPI-obtained data

REAL4Vector *tsChan1 Timeseries data for chan1

REAL4Vector *tsChan2 Timeseries data for chan2

REAL4Vector *dq Data quality information

Container for sending/receiving results between nodes

REAL4Vector *res

Containers for calculated results

`REAL4 variance1` Variance of channel 1 over selected timeseries

`REAL4 variance2` Variance of channel 2 over selected timeseries

`REAL4 *xCorrVal` Cross-correlation values

`REAL4 outputOffShift` Actual time interval between the beginning of the selected GRB and the beginning of the specified 'off'-source segment

`REAL4 badSeconds` The number of 'bad' seconds of data during an 'on'-source event, as determined by the data quality information

`REAL4 expDelay` The expected delay (in seconds) between events at the IFOs, based on the direction of the GRB source; sign is (+) if LHO is first, (-) if LLO is first

`BOOLEAN onSource` A specification whether the selected segment is an 'on'-source (1) or 'off'-source (0)

5.4 Header ExtTrigInitSearch.h

Provides routine to check that the input parameters are reasonalbe, to initialize them, and to allocate global memory.

Synopsis

```
#include "ExtTrigInitSearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"exttrig:Null pointer"
NNUL	2	"exttrig:Non-null pointer"
ARGS	3	"exttrig:Wrong number of arguments"
ALOC	4	"exttrig:Memory allocation error"
NUML	5	"exttrig:Argument is less than a logical limit"
NUMH	6	"exttrig:Argument is greater than a logical limit"
PAR	7	"exttrig:Calling argument has wrong parity (odd)"
SEGZ	8	"exttrig:Number of segments is zero or negative"
RANK	9	"exttrig:Search node has incorrect rank"
UEXT	10	"exttrig:Unrecognised exchange type"
TOTL	12	"exttrig:Total segments exceed given duration"
MASK	13	"exttrig:Inconsistency with initMask parameter"
DUMY	50	"exttrig:Done executing dummy code"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in ExtTrigInitSearch.h on line 1.48.

Structures

struct ETSearchParams

5.5 Header ExtTrigConditionData.h

Breaks the data into segments according to the parameters supplied by the user.

Synopsis

```
#include "ExtTrigConditionData.h"
```

subsection*Error Conditions

<*name*>	code	description
NULL	1	"exttrig:Null pointer"
NNUL	2	"exttrig:Non-null pointer"
MSTR	3	"exttrig:Not search master"
NSEQ	4	"exttrig:Not enough sequences in inPut"
DIMS	5	"exttrig:Wrong dimensions of multiDimData"
NAME	6	"exttrig:Name not found; multiDimData"
FIND	7	"exttrig:multiDimData not found"
INPUT	8	"exttrig:Cannot find data requested"
DATZ	9	"exttrig:Got less data than expected"
TYPE	10	"exttrig:Wrong type of multiDimData"
ALOC	11	"exttrig:Memory allocation error"
ARGS	12	"exttrig:Wrong number of arguments"
TIME	13	"exttrig:gpsRef outside bounds of TSeries"
MPI	14	"exttrig:Invalid MPI request from slave"
FREQ	15	"exttrig:Frequencies are outside bounds"
LGTH	16	"exttrig:Wrong length of multiDimData input"
SPC	17	"exttrig:Wrong spacing within multiDimData"
NOTCH	18	"exttrig:Notch misdefined"
DET	19	"exttrig:Cannot find detector "
DUMY	50	"exttrig:Completed dummy routine"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in ExtTrigConditionData.h on line 1.61.

Structures

5.6 Header ExtTrigApplySearch.h

Executes the external trigger search. This code uses the stochastic code supplied in LAL.

Synopsis

```
#include "ExtTrigApplySearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"exttrig:Null pointer"
NNUL	2	"exttrig:Non-null pointer"
RANK	3	"exttrig:Search node has incorrect rank"
SEND	4	"exttrig:Invalid MPI Message"
ALOC	6	"exttrig:Memory allocation error"
NUMZ	7	"exttrig:Data segment length is zero"
DELT	8	"exttrig:deltaT is zero or negative"
LGTH	9	"exttrig:Different lengths for timeseries"
UEXT	10	"exttrig:Unrecognised exchange type"
DUMY	50	"exttrig:Completed dummy routine"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in ExtTrigApplySearch.h on line 1.42.

Structures

5.7 Header ExtTrigFinalizeSearch.h

Cleans up at the end.

Synopsis

```
#include "ExtTrigFinalizeSearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"exttrig:Null pointer"
NNUL	2	"exttrig:Non-null pointer"
ALOC	3	"exttrig:Memory allocation error"
NUMZ	4	"exttrig:Data segment length is zero"
DELT	8	"exttrig:deltaT is zero or negative"
RANK	9	"exttrig:Search node has incorrect rank"
UEXT	10	"exttrig:Unrecognised exchange type"
DUMY	50	"exttrig:Completed dummy code"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in ExtTrigFinalizeSearch.h on line 1.37.

Structures

References

Chapter 6

Shared object `fct`

This is to illustrate how to run a FCT shared object for `wrapperAPI` under the LDAS.

The input arguments (in your familiar language: filter parameters) of the FCT parallel search code are defined as (N_0,N_{dim}, (start_Indices [1], end_Indices[1], parameter_Segment_Length[1], stride[1]),(start_Indices [2], end_Indices[2], parameter_Segment_Length[2], stride[2]),...(start_Indices [dimension], end_Indices[dimension], parameter_Segment_Length[dimension], stride[dimension])), where N_0 is the number of data points of the input data, N_{dim} is the number of dimension in the parameter (or phase function) space. The rest in the parenthesis are the ranges of each input parameters. For each parameters, you need to have **four** arguments. You should specify (1) the start and (2) end indices, (3) total length of the indices you want the slave to calculate during each call to the ApplySearch, and (4) stride, which is the interval of the indices (within the specified interval at (3)) to be evaluated within each call to the ApplySearch.

All input arguments should be intergers. The convertion from the parameter indices to the physical parameters will be included in the near future. An example schema file can be found in /lalwrapper/example/fct_Lal.schema.

Hope this helps whoever wishes to run the search code.

6.1 Header FCTGeneral.h

Provides ...

Synopsis

```
#include "FCTGeneral.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer."
MEM	2	"Out of memory."

The status codes in the table above are stored in the constants FCTGENERALH_E<*name*>, and the status descriptions in FCTGENERALH_MSGE<*name*>. The source code with these messages is in FCTGeneral.h on line 1.69.

Structures

Chapter 7

Shared object fct-hier

This package is still a (largely undocumented) developer's version. For a more mature code, see the flat-search fct package.

Chapter 8

Shared object inspiral

The inspiral shared object provides functionality to perform a templated binary inspiral search using matched filtering and the χ^2 veto.

The shared object is built around the LAL packages findchirp, bank and inspiral. It contains functionality to allow a search to be performed within the LDAS environment, however this functionality is limited to wrapping the findchirp functions to run within the LDAS environment. The actual search engine and filtering code are part of the LAL findchirp package.

A detailed description of matched filtering as used by findchirp can be found in the LAL Software Documentation; here we document the structure and use of the inspiral shared object.

Users of the inspiral shared object should read the documentation of the -filterparams defined in the header InitSearch.h. This is described in section 8.3.

8.1 Overview

An overview of the operation of the inspiral shared object is as follows. The inspiral shared object should be run in the search master / search slave wrapperAPI paradigm.

1. The command line arguments are parsed and execution execution proceeds based on the contents of the LALInitSearchParams structure.
2. Memory is allocated for the search and the LAL findchirp filtering functions are initialised.
3. An inspiral template parameter bank is created using the functions from the LAL bank package, if required. Otherwise templates parameters are read from the -filterparams command line or from ILWD input files appended to the input by the datacondAPI.
4. The LALConditionData() function parses the input data provided by the wrapperAPI and packs it into structures that can be used by findchirp.
5. The LALApplySearch() function calls FindChirpMaster() on the search master and FindChirpSlave() on the search slaves. On the master, it computes progress information for the wrapperAPI and if InspiralEvents are returned, it calls BuildInspiralOutPut() to convert these structures into the appropriate OutPut structures for the wrapperAPI.
6. Functions from findchirp are called by FindChirpSlave() on the search slaves to perform the inspiral search.
7. Finally, all memory used by the search is deallocated in LALFinalizeSearch().

8.2 Code Flow

In this section, we document the overall algorithm used to execute the inspiral search under LDAS. Sections of the code that are in the inspiral code in lalwrapper are shown in blue text and sections that are in the findchirp code in lal are shown in green text. Sections of the code that are only executed in simulation more are shown in red text.

```
LALInitSearch()
Check the command line parameters passed by the wrapperAPI
Allocate memory for the inspiral parameter structure
Parse the command line arguments
if ( simulation )
      Parse the simulation parameters
Call the findchirp funtions that initialize the filtering code
```

```
if ( templates from bank package )
Call the template bank generation routines to construct the bank
Allocate memory for the input data structures
```

LALConditionData()

```
if ( templates from ilwd )
Parse the input data to construct the bank
Create the template bank
if ( bank minimal match simulation )
      Create a vector to store whether or not the template bank to each slave
Identify the input data and populate the LAL structures by pointing at the input multiDimData
if ( simulation )
      Make a copy of the input data time series
if ( simulation )
      Create the parameters for random number generation
```

LALApplySearch()

```
Initialise the mpi comm world and get the number of slaves
if ( search master )
Call LALFindChirpMaster()
Compute progress information for wrapperAPI
If there are no active slaves, tell the wrapperAPI that the job is finished
if ( search slave )
Call LALFindChirpSlave()
Call BuildInspiralOutPut()
if ( filter output was generated )
Call BuildFilterOutPut()
```

LALFindChirpMaster()

```
while ( the number of active slaves is non-zero )
Wait for a request from a slave
if ( slave requests templates )
if ( bank minimal match simulation )
If the slave has not been sent the template bank to the slave, send the entire template bank to the
slave and remember that that slave has recieved the bank.  If the slave has already been sent a bank
once, tell it to shut down as the simulation is over
If there are any templates in the bank that have not been sent to the slaves, send numTmpltExch
templates to the slave.  If there are no templates left that have not been sent out, tell the slave to
shut down.
if ( slave want to send events )
Master recieves events from the slave.  This can only happen in a bank minimal match simulation.
if ( slave send a number of templates that it has filtered )
Update the progress information for the wrapperAPI.
if ( slave send a finished message )
Decrease the count of active slaves by one.
If the slave has returned events or progress information, we return control to LALApplySearch() so
that it can build the events, calculate the total progress information and return control to the
wrapperAPI.
```

LALFindChirpSlave()

```
Request template parameters from the master
if ( master returns no templates )
Set the flag that tells the wrapperAPI that the search is finished.
while ( 1 )
We loop until we have run out of data.  if ( simulation )
if ( bank minimal match simulation )
Set the response funtion to a constant and the power spectrum to a simulated LIGO I PSD generated by
the noisemodels package.
Create storage for the loudest event over the template bank for each data segment.
Create a random inspiral signal.
Zero the input data and put the generated signal in the middle of the segment.
Call LALFindChirpSPData() to condition the created data.  We need to do this so that we can comute the
normalisation of the injected template.
Compute the normalisation of the injected signal (s|s).

if ( gaussian noise simulation )
Fill the input data with gaussian noise.
Fill the power spectrum with gaussian noise.
```

```
Set the response funtion to a simulated LIGO I response.
  If the data has not been conditioned yet, call LALFindChirpSPData()
Loop over the templates that the master has sent us.
Generate a stationary phase template.
Loop over the data segments.
Call LALFindChirpFilterSegment() to filter the data.  This returns a pointer to a linked list of
InspiralEvents, if events were found, or the NULL pointer if no events were found.
if ( events were found and this is a minimal match simulation )
Store the loudest event found in the bank for each segment.
End loop over data segments.
End loop over templates.
if ( send the events found to the master )
Store the loudest event found in the bank for each segment.
if this is not a simulation break out of the while ( 1 ) loop so control can be returned to
LALApplySearch().
If the simulation counter is non-zero, decrement it by one and return to the top of the while ( 1 )
loop to generate more fake data.  If the simulation counter is zero, so control can be returned to
LALApplySearch().
Tell the master how many templates have been filtered and return control to LALApplySearch()
```

LALFinalizeSearch()

```
Free the memory containing the template bank
Call the findchirp functions that free the memory used by the filtering code
Free the search parameter structure
```

8.3 Header InitSearch.h

Provides LALInitSearch() with a map between -filterparams and search parameters, error messages and a function to read template parameters from the command line.

Synopsis

```
#include "InitSearch.h"
```

This header should only be used by the LALInitSearch() function in the inspiral shared object. It should not be used out of this context.

Command Line Arguments

The number of arguments (including -filterparams) that the inspiral shared object can take is defined in the header file. The number of arguments is determined by the source of the template parameters. The ways to generate the template bank are defined below. In each case the number of command line arguments should be exactly:

```
/* permissable numbers of command line arguments */
#define INSPARG_NBNK    17      /* generate templates using lal bank package */
#define INSPARG_NCMD    18      /* read templates from filterparams          */
#define INSPARG_NILD    16      /* read templates from InPut (ILWD)          */
```

1.58 InitSearch.h

This header also contains the map between position in the -filterparams and the meaning of the command. The shared object performs as strict as possible checking on the command input, but carse should be taken to ensure that appropriate numbers should be given. Here we document the meaning of each of the parameters and give typical values. The #define gives the index of the argv[] array corresponding to to a particular parameter.

```
#define INSPARG_NPTS    1       /* number of points in a segment       */
```

1.78 InitSearch.h

Number of points in a DataSegment. The shared object takes an array of time domain input data and splits it into one or more DataSegment structures which are the basic unit of the findchirp package. More details on the DataSegment segment structure can be found in the LAL documentation for that package. This value must be greater than zero and a power of two. Typical values are 262144–1048576.

```
#define INSPARG_NSEG    2       /* number of data segments             */
```

1.93 InitSearch.h

Number of DataSegment structures to create. This value must be greater than zero and a power of two. Typical values are 1–16.

```
#define INSPARG_OVRL    3       /* data segment overlap                */
```

1.104 InitSearch.h

The overlap between the time domain data in condsecutive DataSegment structures. This value must be ≥ 0. Typical values are 0 or half the length of a data segment. In order to avoid filtering problems, it is suggested at the moment to use half the number of points in a data segment.

```
#define INSPARG_FLOW    4       /* low frequency cutoff                */
```

1.117 InitSearch.h

The low frequency cutoff in Hz used by findchirp. It is suggested to set this to 40.0 for LIGO I. This value must be greater than zero.

```
#define INSPARG_TRNC    5       /* inverse spec trunc                  */
```

1.128 InitSearch.h

The data conditioning function FindChirpSPData() has the ability to truncate the inverse power spectrum in the time domain to remove high Q features in the spectrum. A non-zero value will cause this trunctation to be performed. invSpecTrunc specifies the number of non-zero points in the truncated spectrum in the time domain. Typical values are 0 for no truncation of one quarter of the number of points in a data segment (e.g. $262144/4 = 65536$).

```
#define INSPARG_DYNR    6       /* dynamic range                       */
```

1.144 InitSearch.h

In order to store the strain power spectral density efficently it is necessary to scale the values so that they are within the dynamic range of REAL4. This parameters is $\log_2$ of the dynamic range scaling. Filter output is independent of the parameter. A typical value for LIGO I data is 69.0, so the response function is scaled by $2^{69} \sim 10^{20}$, hence the strain power spectral density is scaled by $\sim 10^{40}$.

```
#define INSPARG_NCHI    7       /* number of chisq bins                */
```

1.159 InitSearch.h

Number of bins for the χ^2 veto perfomed by findchirp. The must be ≥ 0. If set to zero, no χ^2 veto is perfomed. Typical value is 8.

```
#define INSPARG_MTCH    8       /* number of chisq bins                */
```

1.170 InitSearch.h

Minimal match of the template bank used in computing the the χ^2 veto perfomed by findchirp. This must be $0 \leq$ match ≤ 1. Typical value is 0.97.

```
#define INSPARG_MAXC    9       /* maxmimise snr over chirp length     */
```

1.182 InitSearch.h

Maximise over chirp length or not. You decide.

```
#define INSPARG_RHOT    10      /* rhosq threshold                  */
```
1.193 InitSearch.h

A list of two ρ^2 thresholds for the matched filter specified as `(1,2)` where `1` is the threshold for the coarse pass and `2` is the threshold for the fine pass. Values must be positive and chosen appropriately given the input data. If there is no fine bank, the fine threshold is ignored. If the filter output exceeds the threshold, a χ^2 veto is performed, if requested. If not then fine templates are filtered or an event generated as appropriate.

```
#define INSPARG_RHOT    10      /* rhosq threshold                  */
```
1.193 InitSearch.h

A list of two χ^2 thresholds for the veto specified as `(1,2)` where `1` is the threshold for the coarse pass and `2` is the threshold for the fine pass. Values must be positive and chosen appropriately given the input data. If there is no fine bank, the fine threshold is ignored. If the veto is below the threshold for the candidate event then fine templates are filtered or an event generated as appropriate.

```
#define INSPARG_SPEC    12      /* psd estimation type              */
```
1.224 InitSearch.h

Controls how the power spectral density is computed:

- *0* Read PSD from input data (traditional behaviour).
- *1* Compute PSD internally using mean method.
- *2* Compute PSD internally using meadian method.

```
#define INSPARG_NTXC    13      /* number of templates to exchange  */
```
1.239 InitSearch.h

Number of templates to send to each slave per request. This is the number of templates that the master will send to a slave when it requests templates to filter. We are currently determining the most optimum value for the parameter, but the total number of templates divided by the number of slaves is a good place to start from.

```
#define INSPARG_CRVC    14      /* create rhosqVec (or other debugging) */
```
1.253 InitSearch.h

This parameter is known as `create rhosqVec` for historical reasons, but it now controls all the debugging and simulation performed by the shared object. The possible values are:

- `0` No debugging or simulation output.
- `1` Create a vector containing the filter output, $\rho^2(t)$, for the last `DataSegment` filtered against the last template per call to `FindChirpSlave()`. This is stored in the `multiDimData` sent to the `wrapperAPI` and so can be written out as LIGO LW data.
- `2` Tell `FindChirpSlave()` to perform a bank minimal match simulation.

This should be set to 0 unless debugging or simulation is required.

```
#define INSPARG_TSRC    15      /* template source                  */
```
1.275 InitSearch.h

It is possible to obtain the template parameters in three ways. This parmeter determines the method of generation and subsequent parameters in the `-filterparams`. Possible values are:

- `0`: Generate by calls to the lal `bank` package. In this case the bank generation parameters must be specified on the command line, enclosed in parenthesis as the next and final argument (see below). In fact the inspiral shared object calls `LALFindChirpCreateInspiralBank()` which wraps the functions from `bank` so that the template bank is created in the appropriate linked list fashion for use by the search engine.
- `1`: By reading template parameters from the `-filterparams`. This is only suitable for small numbers of templates ≤ 10 and is usuall used for testing and debugging. The number of templates and a list should be specified.
- `2`: By reading template parameters from the `InPut` data. The method is currently only used for debugging. By creating a suitable ILWD file which can be appended to the `wrapperAPI` input by the `datacondAPI`, templates may be passed in along with the input data. No more parameters need to be specified in this case.

```
#define INSPARG_TPRM    16      /* template params                  */
```
1.306 InitSearch.h

If generating templates using the `bank` package then this should be a parenthesis enclosed list of the options passed to the bank generation routines. `numLevel` should be 0 or 1 to generate a flat or a heirarchical template bank respectively. For the definition of the other options see the LAL documentation for the `bank` package:

```
(mMin, MMax, mmCoarse, mmFine, fLower, fUpper, tSampling, iflso, numLevel)
```

If reading template parameters from the `-filterparams`. This should be a single integer specifing the number of templates given on the command line.

```
#define INSPARG_TPRM    16      /* template params                  */
```
1.306 InitSearch.h

If reading template parameters from the `-filterparams`. This should a parenthesis enclosed list of the template parameters mass 1 and mass 2. The masses in each pair should be separated by commas and the pairs separated by forward slashes. Masses should be given in uints of solar mass. For example:

```
(1.4,1.4/2.747,2.325/3.0,4.15)
```

specifies the three templates $m_1 = 1.4, m_2 = 1.4$, $m_1 = 2.747, m_2 = 2.325$ and $m_1 = 3.0, m_2 = 4.15$.

Error codes

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
NUMZ	5	"Data segment length is zero or negative"
SEGZ	6	"Number of data segments is zero or negative"
BINS	7	"Number of chi squared bins is zero or negative"
DTZO	8	"deltaT is zero or negative"
OVLP	9	"Segment overlap is negative"
TRNC	10	"Duration of inverse spectrum in time domain is negative"
FLOW	11	"Inverse spectrum low frequency cutoff is negative"
FREE	12	"Memory free error"
NLAL	13	"Tried to allocate to non-null pointer"
NDAT	14	"No data read"
RHOT	15	"Rhosq threshold error"
CHIT	16	"Chisq threshold error"
DYNR	17	"Dynamic range exceeds range or REAL4"
TMPZ	18	"Number of templates is incorrect"
TMPN	19	"Templates list is null"
TSTR	20	"Error parsing template string"
NTXC	21	"Number of templates in coarse exchange is zero or negative"
TSRC	22	"Invalid template source"
TPRM	22	"Invalid template parameters"
DFLG	23	"Error parsing debug flag"
SPEC	24	"Invalid method of PSD estimation"
BANK	25	"Internal bank generatation is depricated"
MTCH	26	"Invalid template bank minimal match"

The status codes in the table above are stored in the constants INITSEARCHH_E<*name*>, and the status descriptions in INITSEARCHH_MSGE<*name*>. The source code with these messages is in InitSearch.h on line 1.406.

8.4 Command Line Arguments

The `libldastfclusters.so` requires the following arguments to the wrapperAPI:

```
argv[0] = "-filterparams"
[1] = channel name; first two letters must be a valid IFO.
[2] = total number of data in analysis segment
[3] = waveform(s) to inject; for each waveform name X, X_p and X_c must be passed from the datacond API.
[4] = waveform multiplicative amplitude(s)
[5] = right ascension(s) of waveform (rad, between 0 and 2pi)
[6] = declination(s) of waveform (rad, between -pi/2 and pi/2)
[7] = polarization angle(s) (rad)
[8] = number of independent injections to perform (total)
[9] = injection time(s) in number of samples. See notes for details.
[10] = ETG to use (TFCLUSTERS,SLOPE,POWER)
[11] = first ETG parameter
[11+n] = nth ETG parameter
```

8.5 Notes

- Parameters can be numbers, strings, lists, ranges or vectors.
- Parameters can be a list of values in parantheses, for instance (1,2,3,4) or (H1,H2,L1,V,G).
- Numerical values in double parantheses give ranges: ([xlo,xhi,N]) gives N uniformly distributed values between xlo and xhi, ([xlo,xhi,N,r]) gives randomly distributed values (each trial of the simulation correspond to a different realization), ([xlo,xhi,N,l]) gives log distributed values.
- Numerical values without dots or "e" or "E" are assumed to be integers, unless they are in square brackets, where they are assumed to be floats (except for random values, which can be integers).
- String parameters bounded by underscores (e.g. _a_vector_) are assumed to be names of REAL4 sequences passed by the datacondAPI.
- The beginning of the data segment defines the origine of time at the center of the Earth. argv[9] gives the delay between that time and the beginning of the signal time-series, in number of samples, and at the center of the Earth. Given the source position and the interferometer, an additional delay for the wave propagation is added.
- argv[4-7] do not accept arguments of the form ([xlo,xhi,N,r]) with N¿1. When any of these arguments is random and more than one waveform is injected per segment, the same realization of the random argument is used for all injected waveforms. If a list is passed, the list must have the same length for all argv[4-7], and the values of the various arguments will be 'locked' together, i.e. will be used as a single table.
- If argv[5] or argv[6] are outside the allowed range, the waveform is injected with $F_+ = F_\times = 1$ and no time delay with respect to the earth center.
- argv[4-7] and argv[9] can be matrices in order to perform coherent injections for many IFOs. They are passed as ilwd files of dimension (number of segments, injections per segment), using double underscores (e.g. __file__). argv[8] must then be equal to the product of the dimensions of the ilwd file.
- The GW data must be passed with name containing GW_STRAIN_DATA .
- The code uses LALExtractResponse to get the response function.
- The first argument can be set to binOutput while `argv[i] -> argv[i+1]` in order to save the output in a binary format. This leads to up to 2x improvements in speed under LDAS.
- The first or second argument can alternatively be set to fileOutput:path in order to bypass the ligolwAPI completely and speed up large jobs.
- The first or second argument can be set to noLALBurstOutput in order to bypass the burst parameter estimation function. The unaltered output of the ETG is then reported.
- If argv[9] is a list of injection times (esp. random), only those injections for which the signals do not overlap will be performed and reported. The check for overlapping signals is not performed for matrix inputs.
- The first argument can be set to fileOutput:file in order to bypass the LIGOLWAPI and dump data in binary format to file.

8.5.1 Module InitSearchTmplt.c

Parses the template parameters passed using `-filterparams`.

Prototypes

1.49
InitSearchTmplt.c

```
void
InspiralTmpltsFromCmdLine (
    LALStatus                 *status,
    InspiralSearchParams      *params,
    CHAR                     **argv
    )
```

Description

Algorithm

None.

Uses

```
LALCalloc()
LALI4CreateVector()
```

Notes

8.6 Header ConditionData.h

Provides error codes, channel definitions and the InspiralTmpltsFromILWD() function for the shared object ConditionData() function.

Error codes

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
INPT	4	"Wrong data names in InPut structure"
DTZO	5	"deltaT is zero or negative"
NBNK	6	"No template bank has been constructed"
MALF	7	"Malformed ilwd template bank"
CHAN	8	"Incorrect number of input data points"
SPEC	9	"Malformed input power spectrum"
RESP	10	"Malformed input response function"
UIFO	11	"Unknown or malformed IFO name"
DCSP	12	"Redundant spectrum information passed"
MCHN	13	"Multiple data channels found"
MPSD	14	"Multiple power spectra found"
MRSP	15	"Multiple response functions found"
NCHN	16	"No data channel found"
NPSD	17	"No power spectrum found"
NRSP	18	"No response function found"

The status codes in the table above are stored in the constants CONDITIONDATAH_E<*name*>, and the status descriptions in CONDITIONDATAH_MSGE<*name*>. The source code with these messages is in ConditionData.h on line 1.86.

8.7 Header ApplySearch.h

Provides error codes and the BuildInspiralOutPut() function for the shared object ApplySearch() function.

Error codes

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
NUMZ	3	"Data segment length is zero"
DELT	4	"deltaT is zero or negative"
RHOZ	5	"snr squared threshold is zero or negative"
CHIZ	6	"chi squared threshold is zero or negative"
TMPL	7	"linked list of templates to filter in null"
ALOC	8	"Memory allocation error"
RANK	9	"Search node has incorrect rank"
UEXT	10	"Unrecognised exchange type"

The status codes in the table above are stored in the constants APPLYSEARCHH_E<*name*>, and the status descriptions in APPLYSEARCHH_MSGE<*name*>. The source code with these messages is in ApplySearch.h on line 1.71.

8.8 Header FinalizeSearch.h

Provides error codes for the shared object FinalizeSearch() function.

Error codes

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	4	"Memory allocation error"
ARGS	8	"Wrong number of arguments"
NUMZ	16	"Number of data segments is zero"

The status codes in the table above are stored in the constants FINALIZESEARCHH_E<*name*>, and the status descriptions in FINALIZESEARCHH_MSGE<*name*>. The source code with these messages is in FinalizeSearch.h on line 1.56.

Chapter 9

Shared object `load`

9.1 Header Load.h

Provides the LAL-wrapper interface routines for the load shared object.

Synopsis

```
#include <Load.h>
```

Provides the LAL-wrapper interface routines for the load shared object to test load balancing.

Error conditions

<*name*>	code	description
NULL	1	"Null pointer."
NNUL	2	"Non null pointer."
ALOC	4	"Memory allocation error."
ARGS	8	"Incorrect number of arguments."

The status codes in the table above are stored in the constants LOADH_E<*name*>, and the status descriptions in LOADH_MSGE<*name*>. The source code with these messages is in Load.h on line 1.60.

Structures

9.1.1 Module Load.c

The LAL-wrapper interface functions for the load shared object.

Description

This shared object is to test load balancing during a run.

The filterparams input should be a single integer representing the number of requests to send to the slaves. Each request makes the slave do four sleeps. Between each sleep, the slave may be requested to come-up-for-air (i.e., syncronize in the wrapperAPI so that load balancing may be done), and the slave should continue along happily after the load balancing has been done. The master comes-up-for-air frequently and services slave requests (unless it has been directed that load balancing is to be done, in which case it broadcasts a come-up-for-air message to all the slaves).

Algorithm

Uses

Notes

9.1.2 Script LoadTest.sh

Invokes the program happyAPI using mpirun with three processes on the current machine (or the machines specified in the file machines). The program happyAPI is instructed to dynamically load the libtrivial.so shared object. The script LoadTest.sh returns the exit code of happyAPI. The stderr output is redirected to the file Commissar.err while the stdout output is output both to standard output and to the file Commissar.out.

Chapter 10

Shared object power

Functions to implement the excess power search technique which was suggested in Ref. [1] and later independently invented in Ref. [2]. The implementation here is described in detail in Ref. [3]. An intuitive description follows in this section, followed by a detailed explanation required inputs and how to the code.

Flanagan and Hughes [1] (FH) have suggested a particular time-frequency method for blind searches. The method uses only knowledge of the duration and frequency band of the signal: one simply computes the total power within this time-frequency window, and repeats for different start times. The method detects a signal if there is more power than one expects from detector noise alone. Thus, we call it the *excess power* search method. Similar methods have been discussed elsewhere in the gravitational wave literature. Schutz [4] investigated the method in the context of the cross-correlation of outputs from different detectors. An autocorrelation filter for unrestricted frequencies was published by Arnaud *et al.* [5, 6] shortly after and independently of FH. A generalization of the excess power filter has also been discussed in the signal analysis literature [7], where it has recently been "attracting considerable interest" [8]. Finally a method closely related to the excess power method has been explored by Sylvestre [9] and is implemented as the `tfclusters` shared object in LALWrapper.

10.1 Description

The excess power method distinguishes itself for the detection of signals of known duration and frequency band by a single compelling feature: in the absence of any other knowledge about the signal, *the method can be shown to be optimal*. Furthermore, it can be shown that for binary black hole (BBH) mergers of a sufficiently short duration and narrow frequency band, it performs nearly as well as matched filtering.

The essence of the power filter is that one compares the power of the data in the estimated frequency band and for the estimated duration to the known statistical distribution of noise power. It is straightforward to show that if the detector output consists solely of stationary Gaussian noise, the power in the band will follow a χ^2 distribution with the number of degrees of freedom being twice the estimated time-frequency volume (i.e., the product of the time duration and the frequency band of the signal). If a gravitational wave of sufficiently large amplitude is also present in the detector output, an excess of power will be observed; in this case, the power is distributed as a non-central χ^2 distribution [10] with non-centrality parameter given by the signal power. The signal is detectable if the excess power is much greater than the fluctuations in the noise power which scales as the square-root of the time-frequency volume. Thus, the viability of the excess power method depends on the expected duration and bandwidth of the gravitational wave as well as on its intrinsic strength. For instance, the method is not competitive with matched filtering in detecting binary neutron star inspirals, since the time-frequency volume for such signals is very large, $> 10^4$.

To implement this method, one needs to decide the range of frequency bands and durations to search over. For initial LIGO, the most sensitive frequency band is $\sim$ 100–300 Hz, and it makes sense to search just in this band. For binary black hole mergers, signal durations might be of order tens or hundreds of milliseconds, depending on the black hole masses and spins [1]. Thus, the time-frequency volume of a merger signal can be as large as ~ 100, and its power would need to be more than one tenth as large as the noise power for detectability with the excess power method.

One can also establish operational lower bounds on the time durations and frequency bands of interest. Because the largest operational frequency bandwidth is 200 Hz for the initial LIGO interferometers, the shortest duration of signal that need be considered is 5 ms (for a minimum time-frequency volume of unity). Similarly, for a maximum duration of 0.5 s, the smallest bandwidth that needs to be considered is 2 Hz. The excess power in any of the allowed bandwidths and durations can thus be obtained by judiciously summing up power that is output from a bank of one hundred 2 Hz band-pass filters (spanning the 200 Hz of peak interferometer sensitivity) for the required duration.

Having established the statistic and its operational range of parameters, the following simple algorithm for implementing the excess power method emerges naturally:

1. Pick a start time t_s, a time duration δt (containing N data samples), and a frequency band $[f_s, f_s + \delta f]$.
2. Fast Fourier transform (FFT) the block of (time domain) detector data for the chosen duration and start time.

3. Sum the power in each of the $\sim$ one hundred 2 Hz bands spanning the peak sensitivity region of the detector.
4. Further sum the power in the 2 Hz bands which correspond to the chosen frequency band.
5. Calculate the probability of having obtained the summed power from Gaussian noise alone using a χ^2 distribution with $2 \times \delta t \times \delta f$ degrees of freedom.
6. If the probability is significant, record a detection.
7. Repeat the process for all allowable choices of start times t_s, durations δt, starting frequencies f_s and bandwidths δf.

10.2 Command Line Arguments

The command line arguments for the power search code follow. 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 assummed srate $= 2^b$ Hz.
- The frequency band of the signal is between $f_{\rm low}$ Hz and $f_{\rm high}$ Hz. Current versions of the code expect $f_{\rm high} - f_{\rm low} = 2^d$ Hz where d is an integer.
- Minimum time duration,
- Minimum frequency bandwidth.

The libldaspower.so should be used with the following arguments to the wrapperAPI:

argv[0] = "-filterparams"

argv[1] = numPoints Number of data points in a segment is determined by

$$\texttt{numPoints} = (T \times \texttt{srate}) \ .$$

argv[2] = numSegments Number of overlapping segments into which data should be divided for filtering; must be an integer.

argv[3] = ovrlap Number of points overlap between segments. This is an argument for completeness, but in general it should be numPoints/2.

argv[4] = overlapFactor 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.

argv[5] = minFreqBins Smallest extent in frequency of TF tiles to search; must be an integer. A reasonable value for this parameter is 2. The product minFreqBins $\times$ minTimeBins is the minimum time-frequency volume to be searched. See LAL burstsearch package for details.

argv[6] = minTimeBins Smallest extent in time of TF tiles to search; must be an integer. A reasonable value for this parameter is 2. The product minFreqBins $\times$ minTimeBins is the minimum time-frequency volume to be searched. See LAL burstsearch package for details.

argv[7] = flow Lowest frequency in Hz to be searched; a real number. This is obviously $f_{\rm low}$ Hz from our description of the desired signal parameters above.

argv[8] = deltaF This input should be set to $1/T$ Hz but it is ignored by the current version of the code; a real number.

argv[9] = length may be determined by the following formula

$$\texttt{length} = T \times (f_{\rm high} - f_{\rm low}).$$

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

argv[10] = numSigmaMin threshold number of sigma; a real number. Currently see LAL burstsearch package for details.

argv[11] = alphaDefault default alpha value for tiles with sigma ¡ numSigmaMin; a real number. Currently see LAL burstsearch package for details.

argv[12] = segDutyCycle Number of segments sent to slave for analysis; must be an integer. Currently see LAL burstsearch package for details.

argv[13] = alphaThreshold Identify events with alpha less than this; a real number. Currently see LAL burstsearch package for details.

argv[14] = events2Master Number of events to be accepted from a search over numPoints of data; must be an integer.

argv[15] = channelName The name used to identify the data to be analyzed in the multiDimData; a character string matching the channel name in the frame files, e.g. H2:LSC-AS_Q.

argv[16] = simType Type of simulation. Set it to 0, although it is ignored in the current version of the code.

argv[17] = specType Spectrum estimator for whitening data; a character string [useMean, useMedian].

argv[18] = winType 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.]

10.3 Header Power.h

Provides EPSearchParams structure and other things global to the power search.

Synopsis

```
#include "Power.h"
```

Error Conditions

There are no error condition associated with this header file.

Structures

struct EPSearchParams

```
BOOLEAN searchMaster
BOOLEAN haveData
UINT4 *numSlaves
UINT4 currentSegment
INT4 ntotT
INT4 numEvents
UINT4 ovrlap
REAL8 lambda
REAL8 alphaThreshold
EPInitParams *initParams
EPDataSegmentVector *epSegVec
CreateTFTilingIn *tfTilingInput
TFTiling *tfTiling
ComputeExcessPowerIn *compEPInput
```

10.4 Header InitSearch.h

Provides routine to check that the input parameters are reasonable, to initialize them, and to allocate global memory.

Synopsis

```
#include "InitSearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
NUMZ	5	"Data segment length is zero or negative"
SEGZ	6	"Number of data segments is zero or negative"
OVLP	7	"Overlap of data segments is negative"
OVPF	8	"Overlap of TF tiles is negative"
MFBZ	9	"Smallest extent in freq of TF tile is zero"
MTBZ	10	"Smallest extent in time of TF tile is zero"
FLOW	11	"Lowest frequency to be searched is <= 0"
DELF	12	"Freq resolution of 1st time-freq plane is <=0"
LTFZ	13	"Length (Nf) of 1st TF plane (with Nt=1) is <=0"
SIGM	14	"Threshold number of sigma is <=1"
ALPH	15	"Default alpha value is out of range"
FREE	16	"Memory free error"
NLAL	17	"Tried to allocate to non-null pointer"
NDAT	18	"No data read"
DUTY	19	"Number of segments sent to slave is zero"
AMAX	20	"The threshold value of alpha is negative"
E2MS	21	"Number of events out of range[1..99]"
CHNL	22	"Channel name not valid"
SIM	23	"Invalid simulation type: 0, 1, or 2"

The status codes in the table above are stored in the constants INITSEARCHH_E<*name*>, and the status descriptions in INITSEARCHH_MSGE<*name*>. The source code with these messages is in InitSearch.h on line 1.67.

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
NUMZ	5	"Data segment length is zero or negative"
SEGZ	6	"Number of data segments is zero or negative"
OVLP	7	"Overlap of data segments is negative"
OVPF	8	"Overlap of TF tiles is negative"
MFBZ	9	"Smallest extent in freq of TF tile is zero"
MTBZ	10	"Smallest extent in time of TF tile is zero"
FLOW	11	"Lowest frequency to be searched is <= 0"
DELF	12	"Freq resolution of 1st time-freq plane is <=0"
LTFZ	13	"Length (Nf) of 1st TF plane (with Nt=1) is <=0"
SIGM	14	"Threshold number of sigma is <=1"
ALPH	15	"Default alpha value is out of range"
FREE	16	"Memory free error"
NLAL	17	"Tried to allocate to non-null pointer"
NDAT	18	"No data read"
DUTY	19	"Number of segments sent to slave is zero"
AMAX	20	"The threshold value of alpha is negative"
E2MS	21	"Number of events out of range[1..99]"
CHNL	22	"Channel name not valid"
SIM	23	"Invalid simulation type: 0, 1, or 2"

The status codes in the table above are stored in the constants TRACKSEARCHINITH_E<*name*>, and the status descriptions in TRACKSEARCHINITH_MSGE<*name*>. The source code with these messages is in TrackSearchInit.h on line 1.68.

Structures

struct EPSearchParams

10.5 Header ConditionData.h

Breaks the data into segments according to the parameters supplied by the user.

Synopsis

```
#include "ConditionData.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
INPUT	5	"Wrong data names in InPut structure"
DT	6	"Mismatch between deltaT in wrapperParams and inPut"
DTZ	7	"wrapperParams->deltaT is zero or negative"
NUMZ	8	"Data segment length is zero"
SEGZ	9	"Number of data segments is zero"
DATZ	10	"Got less data than expected"
SPEC	11	"Unrecognized power spectrum type"

The status codes in the table above are stored in the constants CONDITIONDATAH_E<*name*>, and the status descriptions in CONDITIONDATAH_MSGE<*name*>. The source code with these messages is in ConditionData.h on line 1.42.

Structures

10.6 Header ApplySearch.h

Executes the excess-power search. This code uses the excess power code supplied in LAL.

Synopsis

```
#include "ApplySearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
NUMZ	4	"Data segment length is zero"
DELT	8	"deltaT is zero or negative"
RANK	9	"Search node has incorrect rank"
UEXT	10	"Unrecognised exchange type"
DELF	11	"Inconsistent deltaF in spectrum and data"

The status codes in the table above are stored in the constants APPLYSEARCHH_E<*name*>, and the status descriptions in APPLYSEARCHH_MSGE<*name*>. The source code with these messages is in ApplySearch.h on line 1.37.

Structures

10.7 Header FinalizeSearch.h

Cleans up at the end.

Synopsis

```
#include "FinalizeSearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	4	"Memory allocation error"
ARGS	8	"Wrong number of arguments"
NUMZ	16	"Number of data segments is zero"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in TrackSearchFinalizeSearch.h on line 1.31.

Structures

References

[1] Éanna É. Flanagan and S. A. Hughes, Phys. Rev. D **57**, 4535-4565 (1998), gr-qc/9701039.

[2] N. Arnaud, F. Cavalier, M. Davier, P. Hello, and T. Pradier, *Triggers for the Detection of Gravitational Wave Bursts*, gr-qc/9903035.

[3] Warren G. Anderson, Patrick R. Brady, Jolien D. E. Creighton and Éanna É. Flanagan, *An excess power statistic for detection of burst sources of gravitational radiation*, PRD 63, 042003 (2001) (gr-qc/0008066). For a short version see *ibid*, *A power filter for the detection of burst sources of gravitational radiation in interferometric detectors*, gr-qc/0001044, submitted to Int. J. Modern. Phys. D.

[4] B. F. Schutz, in *The Detection of Gravitational Waves*, edited by D. G. Blair,(Cambridge University Press, Cambridge, England, 1991), pp. 406–452.

[5] N. Arnaud, F. Cavalier, M. Davier, and P. Hello, Phys. Rev. D **59**, 082002 (1999).

[6] N. Arnaud, F. Cavalier, M. Davier, P. Hello, and T. Pradier, gr-qc/9903025.

[7] J. Fawcett and B. Maranda, IEEE Trans. Inform. Theory **37**, 209 (1991).

[8] R. L. Streit and P. K. Willett, IEEE Trans. Signal Processing **47**, 1823 (1999).

[9] J. Sylvestre, in preparation.

[10] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions* (Dover, New York, 1972).

Chapter 11

Shared object `sick`

This shared object is used to produce dodgy behavior to check how the LALWrapperInterface responds.

11.1 Header Sick.h

Provides the LAL-wrapper interface routines for the sick shared object.

Synopsis

```
#include <Sick.h>
```

Provides the LAL-wrapper interface routines for the sick shared object to check how the LALWrapperInterface routines respond.

Error conditions

<*name*>	code	description
NULL	1	"Null pointer."
NNUL	2	"Non null pointer."
ALOC	4	"Memory allocation error."
ARGS	8	"Incorrect number of arguments."
FAIL	16	"Fail."

The status codes in the table above are stored in the constants SICKH_E<*name*>, and the status descriptions in SICKH_MSGE<*name*>. The source code with these messages is in Sick.h on line 1.62.

11.1.1 Module Sick.c

The LAL-wrapper interface functions for the sick shared object.

Description

The filterparams input should be a single integer representing the illness. The valid illnesses are:

0	ILL_OK	OK
1	ILL_FAIL	Failed LAL routine
2	ILL_RECURSE_FAIL	Recursively failed LAL routine
3	ILL_STAT_DESCRIPT	Status descriptio too long
4	ILL_STAT_LIST	Too many levels in status linked list
5	ILL_LEAK	Memory leak
6	ILL_OUTPUT_OK	Fine output to be freed
7	ILL_OUTPUT_PTR	Output with dodgy pointer
8	ILL_OUTPUT_LIST	Output with dodgy link to middle of list
9	ILL_OUTPUT_TANGLE	Tangled output
10	ILL_SIGSEGV_INIT	SIGSEGV in initSearch
11	ILL_SIGABRT_INIT	SIGABRT in initSearch
12	ILL_SIGSEGV_COND	SIGSEGV in conditionData
13	ILL_SIGABRT_COND	SIGABRT in conditionData
14	ILL_SIGSEGV_APPL	SIGSEGV in applySearch
15	ILL_SIGABRT_APPL	SIGABRT in applySearch
16	ILL_SIGSEGV_FREE	SIGSEGV in finalizeSearch
17	ILL_SIGABRT_FREE	SIGABRT in finalizeSearch
18	ILL_INT	SIGINT somewhere

Algorithm

Uses

Notes

11.1.2 Script SickTest.sh

Invokes the program happyAPI using mpirun with three processes on the current machine (or the machines specified in the file machines). The program happyAPI is instructed to dynamically load the libsick.so shared object. The script SickTestJob.sh returns the exit code of happyAPI. The stderr output is redirected to the file Commissar.err while the stdout output is output both to standard output and to the file Commissar.out. The script SickTest.sh repeatedly runs SickTestJob.sh with illness levels between 0 and 17.

Chapter 12

Shared object simple

This is a shared object that illustrates how to write a shared object for wrapperAPI using the LALwrapper interface.

The simple shared object contains examples of how to create a parameter structure for passing data between the shared object functions, parse input data from the wrapperAPI, create database results and create an ouput multiDimData.

The shared object takes two arguments in its filterparams. The first is the number of points in the input data (must be an integer greater than zero) and the second is the number of times to loop over the LALApplySearch() function.

The simple shared object executes as follows:

- LALInitSearch()
  1. LALInitSearch() checks the sanity of its input parameters and the command line arguments passed to the shared object via the wrapperAPI.
  2. It then allocates memory for the SimpleSearchParams structure, which is used to pass data between functions in the shared object.
  3. After the structure has been created, it parses the command line argumets and places them in the structure for use by the other shared object functions.
  4. It them allocates a REAL4Vector in the parameter strcuture that will be used to pass data from the LALConditionData() function to the LALApplySearch() function.
- LALConditionData()
  1. LALConditionData() checks for the existance of the search parameter structure and casts it from a void pointer to a pointer of type SimpleSerachParams so that it can be defererenced.
  2. The input data passed from the wrapperAPI is then checked to see if it is of the expected type and dimension.
  3. The input data is then put in the form of a LAL data structure. This is done by pointing the data pointer of a REAL4Vector at the data in the multiDimData function, but could be done by creating the vector using LALCreateVector() and then using mmecpy() to copy the data from the input to the LAL structure, as in the inspiral shared object.
  4. Some simple conditioning is then performed on the input data (it is multiplied by -1) and stored in the vector in the parameter structure for use by LALApplySearch()
- LALApplySearch()
  1. LALApplySerach() ignores the input data as it has already been dealt with in the condition data function.
  2. The function checks that the output structure exists and the results pointer is null.
  3. It then checks for the existance of the search parameter structure and casts it from a void pointer to a pointer of type SimpleSerachParams so that it can be defererenced.
  4. A very simple search engine is then executed. If the counter variable in the parameter structure is less than maxCount then counter + 1 fake inspiral events are generated as output.

     The shared object then calculates progress information for the wrapperAPI based on the values of counter and maxCount and increments the value of counter. Control is the returned to the wrapperAPI which handles the output created and then calls LALApplySerach() again.

     The BuildFakeInspiralOutPut() function demonstrates how to build an object for insertion into the meta database.

     If the value of counter equals maxCount then the shared object calls the BuildOptionalOutPut() function which puts the conditioned data into the "optional" element of the output structure. The shared object then sets output->notFinished to zero to inform the wrapperAPI that the search has finished.

- `LALFinalizeSearch()`

  1. `LALFinalizeSearch()` checks for the existance of the search parameter structure and casts it from a void pointer to a pointer of type `SimpleSerachParams` so that it can be defererenced.
  2. It then frees the memory used for the vector of conditioned data.
  3. Finally it frees the SimpleSearchParams structure and returns control to the wrapperAPI.

Chapter 13

Shared object slope

This wrapper runs the slope detectors implemented in the slopefilter package on LIGO data. This package can be found at

${LALHOME}/lal/packages/slopefilters/.

The wrapper code generates triggers for each time at which the peak search algorithm exceeds a fixed threshold. Currently each slope job uses a single CPU of the LDAS system.

13.0.3 Using the wrapper

The wrapper currently permits access to seven slope detection algorithms or 'filters' in the slopefilters library. These algorithms are described in the documentation for this library. The algorithms currently supported are listed in table 13.1. Alongside each filter is an integer specified when using the wrapper to access the library algorithm. The wrapper is called by passing 12 command line arguments that specify the search type, the period (number of bins) over which the slope filter is applied, the threshold for writing a trigger, the separation (in bins) of adjacent triggers, the channel sampling rate, and the channel name. Each command line argument is preceded by a flag starting with a dash that specifies which argument is next. When running the wrapper in standalone, mode, the command line arguments are passed to the .schema file using the -filterparams flag, for example:

```
-filterparams="(-c,ifodmro,-smw,64,-pw,300,-sr,256.0,-t,0.039,-sdtype,2)"
```

The parameters specified after each of the six flages are given in table 13.2. Note that the flag and parameter pairs may be given in any order. The meanings of the different parameters are given in the documentation for the slopefilters package under the LAL documentation.

13.0.4 Using the slope Wrapper in Standalone Mode

Below is a complete boot schema implementing standalone application of the wrapper under MPI on the file wrapper.ilwd.

```
#
# lam boot schema for slope shared object
```

slope parameter	value	filter output
FILTER_OUTPUT_SLOPE	1	fit to slope a_i over N bins
FILTER_OUTPUT_OFFSET	2	fit to offset b_i over N bins
FILTER_OUTPUT_ALF	3	ALF filter (see algorithms)
FILTER_OUTPUT_TAPS_SET_BOXCAR	4	convolve with a uniform window
FILTER_OUTPUT_TAPS_SET_GAUSSIAN	5	convolve with a gaussian window
FILTER_OUTPUT_TAPS_SET_SINE	6	convolve with a period of a sine wave
LALSlopeDetectorFilter()	10	old slope filter, equivalent to FILTER_OUTPUT_SLOPE

Table 13.1: Filters supported by the slope wrapper

flag	parameter	value in example
-c	channel name	ifodmro
-smw	filter period	64 bins
-pw	minimum separation of adjacent triggers	300 bins
-sr	sampling rate	256Hz
-t	trigger threshold	0.039
-sdtype	filter type	2 (FILTER_OUTPUT_SLOPE)

Table 13.2: Filters supported by the slope wrapper

```
#
h -np 2 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-1)" \
-dynlib="/home/edaw/lalcvs/lalwrapper/contrib/slope/src/.libs/libldasslope.so" \
-dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(-c,ifod\
mro,-smw,64,-pw,300,-sr,256.0,-t,0.039,-sdtype,2)" -realTimeRatio=0.9 -doLoadBa\
lance=FALSE -dataDistributor=W -jobID=8 -inputFile="wrapper.ilwd"
```

13.0.5 Using the slope Wrapper in the LDAS Pipeline

Below is an example .tclsh script for launching a slope job under LDAS. Much fancier scripts where the user configurable parameters are set in a short section at the beginning of the script and checked for errors are certainly possible, but this one is intended to indicate what functionality is needed in the script. For more script examples, see the burst stochastic MDC document.

```
#!/ldcg/bin/tclsh

set cmd "
    ldasJob {-name ldas_mdc -password beowulf -email edaw@ligo.mit.edu}
    { dataPipeline
        -dynlib          /ldcg/lib/lalwrapper/libldasslope.so
        -filterparams    (-c,H2:LSC-AS_Q,-smw,64,-pw,300,-sr,16384.0,-t,0.002,-sdtype,2)
        -np              2
        -realtimeratio   0.9
        -datadistributor WRAPPER
        -mddapi          frame
        -datacondtarget  mpi
        -aliases         {gw=_ch0}
        -resultcomment   { PIPELINE01 }
        -resultname      { PIPELINE01 }
        -frames          { /ldas_outgoing/jobs/ldasmdc_data/burst-stochastic/mdc_white_64.F}
-algorithms      { value(gw); }
        -framequery      {Adc(H2:LSC-AS_Q)}
    }"

set sid [ socket ldas.mit.edu 10001 ]
regsub -all -- {[\n\s]+} $cmd { } cmd
puts $sid $cmd
flush $sid
puts [ read $sid ]
close $sid
```

Edward Daw, 18th December 2001

Chapter 14

Shared object stochastic

Implements a stochastic background search using the standard optimally-filtered cross-correlation statistic. This shared object can now handle data from both interferometers and resonant bar detectors, and also either heterodyned or non-heterodyned data streams.

Note: Since no specific mechanism exists for reading the units string from an ILWD into the LALUnit structure, this DSO assumes its inputs have the expected units. These are

- Data Streams $h^{\mathrm{W}}_{1,2}(t)$: ADC counts
- PSDs $P^{\mathrm{W}}_{1,2}(f)$: (ADC counts)^2 / Hz
- Response Functions $\tilde{R}_{1,2}(f)$: ADC counts / attostrain $= 10^{18}$ counts / strain

These are set "by hand" in LALConditionData() without looking at the relevant units fields in the multiDimData. This should be changed to parse the units fields and construct the appropriate LALUnit structures, or at least check to see if it matches one of a set of predefined strings.

Note also that the frequency domain response function expected by the LAL stochastic package routines converts gravitational wave strain into detector output, not the other way around.

14.1 Filter parameters

```
# libldasstochastic.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = baseFreq          Hetrodyning base frequency of time series data
# argv[4]  = windowType1       Index of window type (0=rect, 1=Hann, etc)
# argv[5]  = windowTop1        Width of flat top in middle of window
# argv[6]  = windowType2       Index of window type (0=rect, 1=Hann, etc)
# argv[7]  = windowTop2        Width of flat top in middle of window
# argv[8]  = fullLength        Length after zero-padding
# argv[9]  = numFrequencies    Number of frequencies in optimal filter
# argv[10] = f0                Start frequency for optimal filter
# argv[11] = deltaF            Frequency spacing for optimal filter
# argv[12] = outputNumFreq     Number of frequencies in CC spectrum
# argv[13] = outputF0          Start frequency for CC spectrum
# argv[14] = outputDeltaF      Frequency spacing for CC spectrum
# argv[15] = epochsMatch       Check that time streams are simultaneous
# argv[16] = siteID1           Identifier for first detector geometry
# argv[17] = siteID2           Identifier for second detector geometry
# argv[18] = outputType        Bitmask for DSO outputs
#
```

14.2 Notes

- See the LAL stochastic package for details about stochastic searches using the standard optimally-filtered cross-correlation statistic.
- If `baseFreq` is zero, the shared object expects the input time series to be real, while if it's non-zero, they are assumed to be complex heterodyned time series. Neither the base frequency nor the inittial heterodyning phase is read in from the ILWD input to the wrapper.
- The recognized site identifiers are:

i	Interferometer (differential mode), geometry data specified in input to wrapper
b	Cylindrical bar detector, geometry data specified in input to wrapper
H	LIGO Hanford
L	LIGO Livingston
G	GEO-600
A	ALLEGRO (IGEC orientation)
A###	ALLEGRO with an orientation of ### degrees East (*clockwise*) of North
integer (0-9)	The detector in `lalCachedDetectors[]` with the corresponding index; this is added for backwards compatibility only; the one-letter identifiers are preferred if the geometry has to be specified in the filter parameters and not read from frames.

  The `i` and `b` options should be used under normal circumstances.
- The window type is taken from the `WindowType` enum defined in the `window` package of LAL. It optionally has a flat (windowing function = 1) top of width `windowTop` inserted in the middle. The net effect is, if numPoints $= N$, fullLength $= M$, and windowTop $= P$, to construct a window $\{w_j | j = 0, \ldots, N-P-1\}$ of length $N-P$ and make a zero-padded, windowed data stream according to

$$\bar{h}[k] = \begin{cases} w_k\, h[k] & k = 0, \ldots, \left\lceil \frac{N-P-1}{2} \right\rceil \\ h[k] & k = \left\lceil \frac{N-P-1}{2} \right\rceil + 1, \ldots, \left\lceil \frac{N-P-1}{2} \right\rceil + P \\ w_{P+k}\, h[k] & k = \left\lceil \frac{N-P-1}{2} \right\rceil + P + 1, \ldots, N-1 \\ 0 & k = N, \ldots, M-1 \end{cases} \tag{14.1}$$

- `fullLength` should be at least $2N-1$, although it may be convenient to use $2N$ instead.
- The `outputType` field is a decimal number which describes which combination of outputs from the DSO is desired. The options are

1	Cross-correlation statistic, written to the metadata database
2	Theoretical variance of the cross-correlation statistic, written to the metadata database
4	Cross-correlation spectrum (integrand of the cross-correlation statistic), written to the metadata database
8	Cross-correlation spectrum (integrand of the cross-correlation statistic), written to proc frames

  The options can be combined, so that for instance `outputType=10` means to write the cross-correlation spectrum to frames and the theoretical variance to the database.

14.3 Header Stochastic.h

Provides structures and definitions global to the stochastic search, notably StochasticSearchParams.

Synopsis

```
#include "Stochastic.h"
```

Error conditions

There are no error conditions associated with this header file.

Structures and Unions

Union StochasticFFTPlan

This holds a pointer to an FFTPlan, with the type being determined by whether or not the time series data for the search are heterodyned. The fields are:

RealFFTPlan *realPlan

ComplexFFTPlan *complexPlan

Structure StochasticSearchParams

The fields are:

BOOLEAN searchMaster

BOOLEAN haveData

UINT4 currentSegment

COMPLEX8FrequencySeries *optimalFilter

REAL4WithUnits *sigma

REAL4Vector *window1

REAL4Vector *window2

StochasticFFTPlan fftPlan

StochasticInitParams *initParams

StochasticDataSegmentVector *stochasticSegVec

gpsTimeInterval timeInterval

REAL4 segmentDuration

14.4 Header StochasticData.h

Provides structures needed for StochasticSearchParams, notably including StochasticInitParams, StochasticDataSegment, and StochasticDataSegmentVector.

Synopsis

```
#include "StochasticData.h"
```

Error conditions

<*name*>	code	description
NNUL	1	"Non-null pointer"
NULL	2	"Null pointer"
NSEG	3	"Number of data segments is zero or negative"
NPTS	4	"Number of points in a data segment is zero or negative"
NFRQ	5	"Number of frequencies is zero or negative"
ALOC	6	"Memory allocation error"
STID	7	"Detector Site ID out of range"
UEXT	8	"Unrecognised exchange type"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBData.h on line 1.60.

Structures and Unions

Structure StochasticInitParams

The fields are:

UINT4 numSegments Number of data segments

UINT4 numPoints Number of points in a data segment

REAL8 baseFreq Hetrodyning base frequency of time series data

WindowType windowType1 Type of window to be used on first data stream

UINT4 windowTop1 Width of flat top for first window

WindowType windowType2 Type of window to be used on second data stream

UINT4 windowTop2 Width of flat top for second window

UINT4 fullLength Length after zero-padding

UINT4 numFrequencies Number of frequencies in optimal filter

REAL8 f0 Start frequency for optimal filter

REAL8 deltaF Frequency spacing for optimal filter

UINT4 outputNumFreq Number of frequencies in cross-correlation spectrum

REAL8 outputF0 Start frequency for CC spectrum

REAL8 outputDeltaF Frequency spacing for CC spectrum

BOOLEAN epochsMatch Check that time streams are simultaneous

CHAR siteID1 Identifier for first detector

CHAR siteID2 Identifier for second detector

REAL4 *azimuth1 Azimuth of first detector in degrees East of North (null if not applicable)

REAL4 *azimuth2 Azimuth of second detector in degrees East of North (null if not applicable)

UINT4 outputType Bitmask for DSO outputs

Structure StochasticRealDataSegment

The fields are:

REAL4TimeSeries *data1

REAL4TimeSeries *data2

Structure StochasticComplexDataSegment

The fields are:

COMPLEX8TimeSeries *data1

COMPLEX8TimeSeries *data2

Structure StochasticRealDataSegmentVector

The fields are:

UINT4 length

StochasticRealDataSegment *data

Structure StochasticComplexDataSegmentVector

The fields are:

UINT4 length

StochasticComplexDataSegment *data

Union StochasticDataSegmentVector

The fields are:

StochasticRealDataSegmentVector *realVec

StochasticComplexDataSegmentVector *complexVec

14.4.1 Module StochasticData.c

Routines for creating and destroying a StochasticRealDataSegmentVector or StochasticComplexDataSegmentVector

Prototypes

```
void LALSCreateStochasticDataSegmentVector (
    LALStatus*                      status,
    StochasticRealDataSegmentVector**   vector,
    StochasticInitParams*           params
)
```
1.50 SGWBData.c

```
void LALSDestroyStochasticDataSegmentVector(
    LALStatus*                      status,
    StochasticRealDataSegmentVector**   vector
)
```
1.187 SGWBData.c

```
void LALCCreateStochasticDataSegmentVector (
    LALStatus*                      status,
    StochasticComplexDataSegmentVector**   vector,
    StochasticInitParams*           params
)
```
1.252 SGWBData.c

```
void LALCDestroyStochasticDataSegmentVector(
    LALStatus*                      status,
    StochasticComplexDataSegmentVector**   vector
)
```
1.389 SGWBData.c

Description

Creates and destroys a StochasticRealDataSegmentVector or StochasticComplexDataSegmentVector, which holds the input time-series data for two interferometers.

Algorithms

Uses

- LALMalloc
- LALFree
- LALSCreateVector
- LALSDestroyVector
- LALCCreateVector
- LALCDestroyVector

Notes

```
Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBData.c,v 1.2 2003/09/29 20:11:29 kaice Exp $
```
1.1 SGWBData.c

14.5 Header StochasticInitSearch.h

Header file for a routine that checks the validity of the input parameters, initializes them, and then allocates memory for certain data structures used in the stochastic search.

Synopsis

```
#include "StochasticInitSearch.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
ARG0	5	"Wrong value for argv[0]"
NSEG	6	"Number of data segments is zero or negative"
NODENSEG	7	"Number of segments per nodeis zero or negative"
NPTS	8	"Number of points in a data segment is zero or negative"
BASE	9	"Heterodyning base frequency is negative"
UWIN	10	"Unknown window type"
NWIN	11	"Flat top longer than window"
NFRQ	12	"Number of frequencies is zero or negative"
F0	13	"Start frequency is negative"
DELF	14	"Frequency spacing is zero or negative"
DTSZ	15	"Nonstandard size of COMPLEX8 and/or UCHAR"
FRES	16	"Frequency sampling for output finer than for optimal filter"
FRNG	17	"Frequency range for output not fully contained in optimal filter"
FNUM	18	"Number of frequencies in optimal filter is greater than number of data elements"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBInitSearch.h on line 1.80.

Structures

None.

14.5.1 Module StochasticInitSearch.c

Routine that checks the validity of the input parameters, initializes them, and then allocates memory for certain data structures used in the stochastic search.

Prototypes

1.56
SGWBInitSearch.c

```
void LALInitSearch(
    LALStatus*              status,
    void**                  searchParams,
    LALInitSearchParams*    initSearchParams
)
```

Description

Checks the validity of the input parameters, initializes them, and then allocates memory for the time-series input data, optimal filter, and windowing functions.

Algorithms

Uses

- LALMalloc
- LALCCreateVector
- LALCreateForwardRealFFTPlan
- LALCreateStochasticDataSegmentVector

Notes

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBInitSearch.c,v 1.7 2003/10/07 17:17:29 jolien Exp $

14.6 Header StochasticConditionData.h

Header file for a routine that does one-time processing in preparation for the stochastic search.

Synopsis

```
#include "StochasticConditionData.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"
MSTR	2	"Not search master"
NSEQ	3	"Wrong number of sequences in InPut structure"
TYPE	4	"Input data has wrong type"
DIMS	5	"Input data has wrong dimensions"
SIZE	6	"Input data has wrong size"
NAME	7	"Wrong data name in InPut structure"
ALOC	8	"Memory allocation error"
UNIT	9	"Input data had unexpected units"
STID	10	"Detector site identifier unrecognized"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBConditionData.h on line 1.61.

Structures

None.

14.6.1 Module StochasticConditionData.c

Routine for performing one-time processing in preparation for the stochastic search.

Prototypes

1.131
SGWBConditionData.c

```
void LALConditionData (
                       LALStatus*          status,
                       LALSearchInput*     inout,
                       void*               searchParams,
                       LALMPIParams          *mpiParams
                       )
```

Description

Performs one-time processing (e.g., extraction of input data, construction of windowing functions, and calculation of the optimal filter for a stochastic background having $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$) in preparation for the stochastic search.

Algorithms

LALConditionData does the following:

1. Constructs the windowing functions requested for the two data streams
2. Checks the validity of the input data, and extracts the detector geometry, time-series data, power spectral densities, and instrument response functions from the multidim input data.
3. Converts 2-sided power spectral densities (produced by the datacondAPI) to 1-sided power spectra having the appropriate bandwidth and frequency resolution for construction of the optimal filter.
4. Coarse-grains the instrument response functions to the appropriate bandwidth and frequency resolution for construction of the optimal filter.
5. Casts the extracted input data to types expected by the LAL stochastic routines.
6. Calculates the overlap reduction function given the detector geometry information.
7. Sets $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$ for this particular search.
8. Calculates inverse calibrated and half-calibrated power spectra given the calibrated power spectra and instrument response functions.
9. Calculates the optimal filter for a stochastic background having $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$ using the previously calculated information.

Uses

- LALMalloc
- LALFree
- LALCreateVector
- LALCCreateVector
- LALSCreateVector
- LALDestroyVector
- LALCDestroyVector
- LALSDestroyVector
- LALSCoarseGrainFrequencySeries
- LALCCoarseGrainFrequencySeries
- LALStochasticOmegaGW
- LALStochasticInverseNoise
- LALStochasticOptimalFilter
- LALStochasticOptimalFilterNormalization
- CastToReal4
- CastToComplex8

Notes

- LALConditionData calls two static functions CastToReal4 and CastToComplex8, which are used to cast time-series data, power spectral densities, and instrument response functions to the proper type for the LAL stochastic routines.
- Since no specific mechanism exists for reading the units string from an ILWD into the LALUnit structure, this DSO assumes its inputs have the expected units. These are
  - Data Streams $h^{\mathrm{W}}_{1,2}(t)$: ADC counts
  - PSDs $P^{\mathrm{W}}_{1,2}(f)$: (ADC counts)^2 / Hz
  - Response functions $\tilde{R}_{1,2}(f)$: ADC counts / attostrain $= 10^{18}$ counts / strain

  These are set "by hand" in LALConditionData() without looking at the relevant units fields in the multiDimData. This should be changed to parse the units fields and construct the appropriate LALUnit structures, or at least check to see if it matches one of a set of predefined strings.

14.7 Header SGWBApplySearch.h

Header file for a routine that implements the stochastic background search using the standard optimally-filtered cross-correlation statistic.

Synopsis

```
#include "SGWBApplySearch.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
MSTR	3	"Search master not being used"
RANK	4	"Search master has incorrect rank"
ALOC	5	"Memory allocation error"
SIZE	104	"Invalid data length"
ROWS	120	"Invalid number of rows"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBApplySearch.h on line 1.54.

Structures

struct StochasticSearchOutput

The fields are:

REAL4WithUnits *ccStatReal

COMPLEX8WithUnits *ccStatComplex

REAL4WithUnits *sigma

COMPLEX8FrequencySeries *ccSpec

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBApplySearch.h,v 1.2 2003/09/03 02:37:39 kaice Exp $

14.7.1 Module StochasticApplySearch.c

Routine that implements the stochastic background search, calculating the values of the spectrum of the optimally-filtered cross-correlation statistic, building the output for subsequent insertion into the metadata database and frames.

Prototypes

1.101 SGWBApplySearch.c

```
void LALApplySearch(
                    LALStatus             *status,
                    LALSearchOutput       *output,
                    LALSearchInput        *input,
                    LALApplySearchParams  *applyParams
                    )
```

Description

Implements the search for a stochastic background, calculating the values of the spectrum of the optimally-filtered cross-correlation statistic with $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$. The cross-correlation statistic spectrum values are then built for subsequent insertion into the metadata database and frames using the StochasticBuildOutput routine.

Algorithms

Given time-series data from two interferometers (pre-processed by the datacondAPI) and values of the optimal filter (calculated in LALConditionData), LALApplySearch does the following:

1. Splits the input time-series data into numSegments≥ 10 segments, each of length numPoints.
2. Windows, zero-pads, and FFTs each segment of time-series data.
3. Calculates the values of the standard optimally-filtered cross-correlation statistic for each segment of data, using the values of the optimal filter previously calculated in LALConditionData.
4. Coarse-grains the cross-correlation statistic spectrum output to the desired bandwidth and frequency resolution.
5. Builds the cross-correlation statistic spectrum output (for each data segment) for subsequent insertion into the metadata database and frames.

Uses

- LALCCreateVector
- LALSZeroPadAndFFT
- LALStochasticCrossCorrelationSpectrum
- LALCCoarseGrainFrequencySeries
- StochasticBuildOutput
- LALCDestroyVector

Notes

The analysis is run on *only* the search master node of the beowulf, since we are currently considering the simple case of stochastic signals with $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$. In the future, searches for stochastic backgrounds having different spectra (e.g., power law $\Omega_{\rm gw}(f) \propto f^{\alpha}$) should be run on multiple slaves (e.g., one node for each α).

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBApplySearch.c,v 1.7 2003/10/07 17:17:09 jolien Exp $

1.1 SGWBApplySearch.c

14.7.2 Module StochasticBuildOutput.c

Routine that builds the output of the optimally-filtered cross-correlation statistic spectrum for insertion into the metadata database and frames.

Prototypes

1.70 SGWBBuildOutput.c

```
void StochasticBuildOutput (
      LALStatus*                 status,
      LALSearchOutput*           output,
      StochasticSearchOutput*    input,
      StochasticSearchParams    *params
)
```

1.637 SGWBBuildOutput.c

```
void
LALCreateDBEntryBLOBU(
    LALStatus *status,
    dataBase  *output,
    LALBLOB   *blob,
    UINT4      rows
    )
```

Description

StochasticBuildOutput() builds the output of the optimally-filtered cross-correlation statistic spectrum for subsequent insertion into the metadata database (as a summ_spectrum table) and frames (as a sequence in filter_output).

LALCreateSearchSummvarsDatabase() creates a search_summvars database entry, analogous to LALCreateSearchSummaryDatabase(), while is a version of which treats the BLOB as (unsigned char *) rather than (signed char *).

Algorithms

Uses

- LALCreateSearchSummaryDataBase
- LALCreateSearchSummvarsDataBase
- LALCHARCreateVector
- UnitAsString
- LALCHARDestroyVector
- LALCreateNextDBEntry
- LALCreateDBEntryData
- LALCreateDBEntryBLOB
- LALCreateDBEntryBLOBU
- LALMalloc
- LALCalloc

Notes

1.1 SGWBBuildOutput.c

```
Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBBuildOutput.c,v 1.3 2003/09/03 02:37:17 kaice Exp $
```

14.8 Header StochasticFinalizeSearch.h

Header file for a routine that cleans up after the stochastic search.

Synopsis

```
#include "StochasticFinalizeSearch.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBFinalizeSearch.h on line l.46.

Structures

None.

14.8.1 Module StochasticFinalizeSearch.c

Routine that cleans up after the stochastic search.

Prototypes

1.51
SGWBFinalizeSearch.c

```
void LALFinalizeSearch(
    LALStatus*          status,
    void**              searchParams
)
```

Description

Cleans up after the stochastic search–i.e., frees memory previously allocated in the InitSearch routine.

Algorithms

Uses

- LALSDestroyStochasticDataSegmentVector
- LALCDestroyStochasticDataSegmentVector
- LALCDestroyVector
- LALFree
- LALDestroyRealFFTPlan
- LALDestroyComplexFFTPlan

Notes

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBFinalizeSearch.c,v 1.4 2003/09/03 02:37:17 kaice Exp $

1.1
SGWBFinalizeSearch.c

Chapter 15

Shared object tfclusters

Implementation of time-frequency transient detection, with power thresholds and clustering analysis. For details about the algorithm, see the package tfclusters in the LAL library.

15.1 Description

The time series of N_{total} data points (sampled at frequency f_s) is first splitted into n_{nodes} segments of N_{seg} data points, where n_{nodes} is usually taken to be the number of nodes available for computation. The segments should overlap by N_{overlap} samples, where N_{overlap}/f_s is at least as large as the longest expected burst in the data. For every segment, the analysis is performed with the arguments set as explained in the LAL library documentation. The output of every one of these analyses is a list of detected bursts. The last step is the merging of all these lists (which is done on the serach master node), in order to remove duplicated bursts.

15.2 Command Line Arguments

The libldastfclusters.so requires the following arguments to the wrapperAPI:

argv[0] = "-filterparams"

argv[1] = channel name; if SNGL_BURST, first two letters go into IFO. If GDS_TRIGGER, full name goes into subtype.

argv[2] = metadata database table (GDS_TRIGGER or SNGL_BURST)

argv[3] = total number of data (N_{total})

argv[4] = number of samples in one segment (N_{seg})

argv[5] = time resolution in seconds* (T)

argv[6] = sampling frequency (f_s)

argv[7] = overlap between segments (number of samples) (N_{overlap})

argv[8] = black pixel probability*

argv[9] = absolute error goal on power threshld (Set to <0 for white noise of unit variance at input)*. Ignored if a spectrum is passed by datacondAPI. If a spectrum is passed, it must be from DC to Nyquist inclusively, and have a resolution 1/argv[5]. It should use a rectangular window, or else the black pixel probability will be different from argv[8].

argv[10] = α^*

argv[11] = minimum frequency to process (Hz)*

argv[12] = maximum frequency to process (Hz)*

argv[13] = number of standard deviations above estimated mean to reject in second pass at mean and variance estimation

argv[14] = σ^*

argv[15] = distance for s1 = 1, s2 = 1, i.e. $\delta^*_{s_1,s_2}$

argv[16] = distance for s1 = 1, s2 = 2*

...

argv[$15 + \sigma(\sigma - 1)$] = distance for s1 = σ-1, s2 = σ-1*

argv[$16 + \sigma(\sigma - 1)$] : if this argument is present, writes pixel coordinates to database table sngl_transdata. Actual value is unimportant.

Note: * refers to arguments directly related to the search algorithm, and described in the LAL Library documentation.

15.3 Special Notes:

- $\frac{N_{\text{total}} - N_{\text{overlap}}}{n_{\text{nodes}}} + N_{\text{overlap}} = N_{\text{seg}}$ must always be respected.
- N_{seg}/f_s must be a positive integer.

15.4 Header InitSearch.h

Provides routine to check that the input parameters are reasonable, to initialize them, and to allocate global memory.

Synopsis

```
#include "InitSearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
SPOS	5	"Parameter <= 0 expected to be strictly positive"
NAN	6	"Numerical parameter is NaN"
AOR	7	"Parameter expected to be in [0,1] is out of range"
POS	8	"Parameter < 0 when expected to be positive or zero"
MMF	9	"Parameter minf > maxf"
NAM	10	"Channel/IFO name is zero length"
TAB	11	"Unknown table name"
IN	12	"invalid input"
RANK	13	"rank of node"
MAXBLOB	14	"Maximum allowed BLOB size (1 Mo) exceeded"
ETG	15	"ETG function generated a SIGSEGV or SIGBUS signal!"
ETGDATA	16	"ETG function modifies its input!"
NB	17	"Maximum number of burst exceeded!"
UI	18	"Unimplemented feature!"
FILE	19	"File error!"

The status codes in the table above are stored in the constants INITSEARCHH_E<*name*>, and the status descriptions in INITSEARCHH_MSGE<*name*>. The source code with these messages is in InitSearch.h on line 1.206.

Structures

struct TFCSearchParams

BOOLEAN searchMaster set to 1 if search master

UINT4 NData total number of data to be searched

UINT4 Ttotal total number of data in one segment

REAL8 T time resolution

UINT4 fs sampling frequency of channel

UINT4 olap overlap (# of samples) between segments

REAL4 bpp black pixel probability

REAL4 eGoal absolute error goal on power threshld

REAL4* dmro container for the raw dataset

CListDir* dir container for the instructions

REAL4TimeSeries* tseries container for the data to be analyzed by each node

TFPlaneParams* tspec container for the time-frequency parameters

DbTable table database table name

char name[256] channel name

REAL4* resp response function

REAL4* spec power spectrum

15.5 Header ConditionData.h

Breaks the data into segments according to the parameters supplied by the user.

Synopsis

```
#include "ConditionData.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
INPUT	5	"Wrong data names in InPut structure"
DT	6	"Mismatch between deltaT in wrapperParams and inPut"
DTZ	7	"wrapperParams->deltaT is zero or negative"
NUMZ	8	"Data segment length is zero"
SEGZ	9	"Number of data segments is zero"
DATZ	10	"Got less data than expected"
NDAT	11	"Number of power thresholds doesn't match Nyquist frequency of channel"
DTYP	12	"Datatype not supported"
DMRO	13	"Missing data for requested channel"
ISIG	14	"Invalid injected signal"
ISIGO	15	"Injected signal has invalid offset"
PEST	16	"Unable to fit to Rice"
ZERO	17	"Zero size time-series"
RICE	18	"Error in RICE algorithm"
IFO	19	"Unknown IFO name"
NIFO	20	"IFO in conditionData not passed as argument"
IFIL	21	"Invalid filter"
NDS	22	"Wrong number of sequences passed"
CMAT	23	"Invalid correlation matrix"

The status codes in the table above are stored in the constants CONDITIONDATAH_E<*name*>, and the status descriptions in CONDITIONDATAH_MSGE<*name*>. The source code with these messages is in ConditionData.h on line 1.66.

15.6 Header ApplySearch.h

Executes the excess-power search. This code uses the excess power code supplied in LAL.

Synopsis

```
#include "ApplySearch.h"
```

Error Conditions

<*name*>	code	description
NULL	11	"Null pointer"
NNUL	12	"Non-null pointer"
ALOC	13	"Memory allocation error"
NUMZ	14	"Data segment length is zero"
DELT	18	"deltaT is zero or negative"
RANK	19	"Search node has incorrect rank"
UEXT	110	"Unrecognised exchange type"
DELF	111	"Inconsistent deltaF in spectrum and data"
NDAT	111	"Total number of data not a multiple of (segment length - overlap)"
IERR	112	"Internal error"
THREAD	113	"Thread error"
MAXBLOB	114	"Maximum allowed BLOB size (! Mo) exceeded"

The status codes in the table above are stored in the constants APPLYSEARCHH_E<*name*>, and the status descriptions in APPLYSEARCHH_MSGE<*name*>. The source code with these messages is in ApplySearch.h on line 1.49.

15.7 Header FinalizeSearch.h

Cleans up at the end.

Synopsis

```
#include "FinalizeSearch.h"
```

Error Conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	4	"Memory allocation error"
ARGS	8	"Wrong number of arguments"
NUMZ	16	"Number of data segments is zero"

The status codes in the table above are stored in the constants FINALIZESEARCHH_E<*name*>, and the status descriptions in FINALIZESEARCHH_MSGE<*name*>. The source code with these messages is in FinalizeSearch.h on line 1.31.

Chapter 16

Shared object `trivial`

This is a shared object that is used to debug the LALWrapper interface. It should not be taken as a guide to writing shared objects, as it does several things that a serach shared object should not do, such as write to standard output.

All that this shared object does is to print the song "Swingin' On A Star." For an example of how to write a search shared object, see the `simple` shared object.

Would you like to swing on a star,
Carry moonbeams home in a jar,
And be better off than you are,
Or would you rather be a mule?

A mule is an animal with long funny ears,
Kicks up at anything he hears,
His back is brawny, but his brain is weak,
He's just plain stupid with a stubborn streak,
And by the way, if you hate to go to school,
You may grow up to be a mule!

Would you like to swing on a star,
Carry moonbeams home in a jar,
And be better off than you are,
Or would you rather be a pig?

A pig is an animal with dirt on his face,
His shoes are a terrible disgrace,
He has no manners when he eats his food,
He's fat 'n lazy and extremely rude,
But if you don't care a feather or a fig,
You may grow up to be a pig!

Would you like to swing on a star,
Carry moonbeams home in a jar,
And be better off than you are,
Or would you rather be a fish?

A fish won't do anything but swim in a brook,
He can't write his name or read a book,
To fool the people is his only thought,
And though he's slippery, he still gets caught,
But then if that sort of life is what you wish,
You may grow up to be a fish!

And all the monkeys aren't in the zoo,
Everyday, you'll meet quite a few,
So, you see, it's all up to you,
You can be better than you are,
You could be swingin' on a star!

16.1 Header Trivial.h

Provides the LAL-wrapper interface routines for the Trivial shared object.

Synopsis

```
#include <Trivial.h>
```

Provides the LAL-wrapper interface routines for the Trivial shared object.

Error conditions

<*name*>	code	description
NULL	1	"Null pointer."
NNUL	2	"Non null pointer."
ALOC	4	"Memory allocation error."
PUTS	8	"Error in puts()."

The status codes in the table above are stored in the constants TRIVIALH_E<*name*>, and the status descriptions in TRIVIALH_MSGE<*name*>. The source code with these messages is in Trivial.h on line 1.60.

Structures

```
enum { Mule, Pig, Fish, Finale, NumVerse }
struct TrivialParams
```

This structure stores the verses, labelled Mule, Pig, Fish, and Finale, of the song. The fields are:

`const char *verse[NumVerse]` The array of verses in the song.

16.1.1 Module Trivial.c

The LAL-wrapper interface functions for the Trivial shared object.

Description

The routines LALInitSearch(), LALConditionData(), LALApplySearch(), and LALFinializeSearch() do nothing more than print the four verses of the song.

Algorithm

Uses

Notes

16.1.2 Script TrivialTest.sh

Invokes the program happyAPI using mpirun with three processes on the current machine (or the machines specified in the file machines). The program happyAPI is instructed to dynamically load the libtrivial.so shared object. The script TrivialTest.sh returns the exit code of happyAPI. The stderr output is redirected to the file Commissar.err while the stdout output is output both to standard output and to the file Commissar.out.

Chapter 17

Shared object waveburst

17.1 Description

Waveburst DSO is used to search for coincident bursts in a pair of interferometers at a time. It can also be applied to a single interferometer.

Two **AS_Q** channels are taken as an input. Wavelet transform is applied to each channel to represent data as a set of wavelet coefficients indexed by time and frequency. Percentile transform selects N% of these coefficients (pixels in two-dimensional time-frequency plot) with the greatest absolute value. All the other pixels are zeroed out. Non-zero coefficients are replaced by percentile amplitudes (the bigger the percentile amplitude, the bigger the original coefficient) in order to be able later to compute correlation between two events. The resulting 2D matrices are ANDed to zero out for each interferometer pixels that are not in time-frequency coincidence in both interferometers (the chosen time resolution is comparable with 10ms and therefore no time shift is required to apply coincidence if we use waveburst for **LLO** and **LHO**). Clustering is applied to the remaining non-zero pixels in each interferometer to further select only those burst events that satisfy certain criteria on cluster size, power, correlation, likelihood, etc. Finally, cross-correlation between the corresponding clusters in two interferometers is computed to estimate how similar are the two events.

The results for each channel are written to **waveburst** and **waveburst_mime** tables in the database. **waveburst** table records the characteristics of the burst events (cluster size, correlation, power, likelihood, etc). **waveburst_mime** table records percentile and original amplitudes in the smallest rectangle that contains the cluster; that allows to restore the waveform of the signal if needed. As required, waveburst DSO also makes a record to **search_summary** table.

17.2 Input parameters

OPTION	DESCRIPTION	DEFAULT
-ch1	channel one	none
-ch2	channel two	none
-WaveletType	wavelet type	BIORTHOGONAL
-WaveletTreeType	wavelet tree type	BINARY
-WaveletLevel	wavelet decomposition level	5
-WaveletBorder	wavelet border type	B_POLYNOM
-WaveletHPFilterLength	wavelet highpass filter length	10
-WaveletLPFilterLength	wavelet lowpass filter length	10
-nonZeroFraction	fraction of pixels that remain non-zero after percentile transform	0.3
-timeWindowNanoSec	maximum time window difference between events in two ifos	0
-halo	this flag indicates whether we use halo or not during clustering	0
-minClusterSize	minimum allowed size of clusters (smaller clusters are zeroed out)	4
-maxClusterSize	maximum allowed size of clusters (not used at the moment)	∞
-likelihoodThreshold	minimum likelihood of the cluster (clusters that do not satisfy this criteria are not written to the database	0
-correlationThreshold	minimum correlation between pixels of the same cluster	0
-rcorrelationThreshold	minimum correlation between corresponding clusters in two channels	0
-simulationType	type of simulation (not used yet)	0
-pixelMixer	the flag is 1 if pixel mixer is used and 0 otherwise	0
-pixelSwapOne	the flag is 1 if pixels are swapped in time on each wavelet layer in channel one	0
-pixelSwapTwo	the flag is 1 if pixels are swapped in time on each wavelet layer in channel two	0
-o1	debugging flag: dump the original wavelet series in channel one to the file if 1	0
-o2	debugging flag: dump the original wavelet series in channel two to the file if 1	0
-p1	debugging flag: dump the wavelet series after percentile transform in channel one to the file if 1	0
-p2	debugging flag: dump the wavelet series after percentile transform in channel two to the file if 1	0
-co1	debugging flag: dump the wavelet series after after coincidence in channel one to the file if 1	0
-co2	debugging flag: dump the wavelet series after after coincidence in channel two to the file if 1	0
-cl1	debugging flag: dump the wavelet series after after clustering in channel one to the file if 1	0
-cl2	debugging flag: dump the wavelet series after after clustering in channel two to the file if 1	0
-outputToFile	debugging flag: possible values are PT (percentile), PS (pixel swap), PM (pixel mixing); depending on this value the above debugging flag either apply to the original data, data to which pixel swap or pixel mixing was applied	PT
-outputBlobs	if 1, blobs should be put into the database	0

17.3 Sample tcl script

Here is the simplest script to run waveburst DSO. The script file is called **wave.tcl**, the parameter file is called **params.dat**. To run a job, execute
wave.tcl params.dat.

17.3.1 Script file

```
package require LDASJob

if { $argc!=1 } {
    puts "$argv0 paramsFile";
    exit
}

set paramsFile [lindex $argv 0]
source $paramsFile

set endIndex [ expr int($duration*$srate)-1 ]
set length [ expr $endIndex+1 ]

set etime [ expr $stime + $duration - 1 ]
set times $stime-$etime

LJrun job1 -manager $manager {
dataPipeline
   -subject { $subject }
   -dynlib $dso
   -filterparams (-ch1,$channel1,-ch2,$channel2,-WaveletType,$fp_wtype,-WaveletTreeType,
$fp_wttype,-WaveletLevel,$fp_wl,-WaveletBorder,$fp_wb,-WaveletHPFilterLength,
$fp_whpfl,-WaveletLPFilterLength,$fp_wlpfl,-nonZeroFraction,$fp_nzf,
-timeWindowNanoSec,$fp_twns,-halo,$fp_h,-minClusterSize,$fp_mincs,-maxClusterSize,
$fp_maxcs,-likelihoodThreshold,$fp_lt,-correlationThreshold,$fp_ct,-simulationType,
$fp_stype,-pixelMixer,$fp_pm,-pixelSwapOne,$fp_ps1,-pixelSwapTwo,$fp_ps2,-o1,
$fp_o1,-o2,$fp_o2,-p1,$fp_p1,-p2,$fp_p2,-co1,$fp_co1,-co2,$fp_co2,-cl1,
$fp_cl1,-cl2,$fp_cl2,-outputToFile,$fp_o2f)
   -np 2
   -realtimeratio 0.9
   -framequery { { $frameType1 { $ifo1 } {} $times  Adc($channel1) }
{ $frameType2 { $ifo2 } {} $times  Adc($channel2) } }
   -datacondtarget wrapper
   -metadataapi ligolw
   -metadatatarget datacond
   -multidimdatatarget { ligolw }
   -database { $db }
   -aliases { l1=$al1; h2=$al2;}

   -algorithms {
       w = Biorthogonal($fp_whpfl, 1);

       l1w = WaveletForward( l1, w, $fp_wl );
       l1ws=getSequence(l1w);
       l1wts=tseries(l1ws,$srate,$stime);
       output(l1wts, ,wrapper,l1wts, l1wts);

       l1w = WaveletForward( h2, w, $fp_wl );
       l1ws=getSequence(l1w);
       l1wts=tseries(l1ws,$srate,$stime);
       output(l1wts,,wrapper, h2wts, h2wts);

   }
}
```

17.3.2 Parameters file

```
set manager MIT
set db mit_test
set subject "Testing wave DSO"

set channel2 H2:LSC-AS_Q
set channel1 L1:LSC-AS_Q

set frameType1 RDS_R_L2
set frameType2 RDS_R_L2

set ifo1 L
set ifo2 H

set al1 L1
set al2 H2

set stime 714189824
set duration 1

set srate 16384.0
set dso libldaswave.so

set fp_wtype 1
set fp_wttype 1
set fp_wl 5
set fp_wb 4
set fp_whpfl 10
set fp_wlpfl 10
set fp_nzf 0.2
set fp_twns 0
set fp_h 0
set fp_mincs 4
set fp_maxcs 1000
set fp_lt 0.0
set fp_ct 0.0
set fp_stype 0
set fp_pm 0
set fp_ps1 0
set fp_ps2 0
set fp_o1 0
set fp_o2 0
set fp_p1 0
set fp_p2 0
set fp_co1 1
set fp_co2 0
set fp_cl1 1
set fp_cl2 0
set fp_o2f PT
```

17.4 Header Wave.h

Provides the LAL-wrapper interface routines for the waveburst shared object.

Synopsis

```
#include <Wave.h>
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer."
NSEQ	2	"Number of sequences is not 2"
UNITS	3	"Inconsistent units"
TYPES	5	"Inconsistent data types"
SPACES	6	"Inconsistent sequence type"
DIMS	7	"Inconsistent dimensions"

The status codes in the table above are stored in the constants WAVEH_E<*name*>, and the status descriptions in WAVEH_MSGE<*name*>. The source code with these messages is in Wave.h on line 1.58.

Chapter 18

Shared object `burstwrapper`

18.1 Command Line Arguments

The libldastfclusters.so requires the following arguments to the wrapperAPI:

```
argv[0] = "-filterparams"
[1] = channel name; first two letters must be a valid IFO.
[2] = total number of data in analysis segment
[3] = waveform(s) to inject; for each waveform name X, X_p and X_c must be passed from the datacond API.
[4] = waveform multiplicative amplitude(s)
[5] = right ascension(s) of waveform (rad, between 0 and 2pi)
[6] = declination(s) of waveform (rad, between -pi/2 and pi/2)
[7] = polarization angle(s) (rad)
[8] = number of independent injections to perform (total)
[9] = injection time(s) in number of samples. See notes for details.
[10] = ETG to use (TFCLUSTERS,SLOPE,POWER)
[11] = first ETG parameter
[11+n] = nth ETG parameter
```

18.2 Notes

- Parameters can be numbers, strings, lists, ranges or vectors.
- Parameters can be a list of values in parantheses, for instance (1,2,3,4) or (H1,H2,L1,V,G).
- Numerical values in double parantheses give ranges: ([xlo,xhi,N]) gives N uniformly distributed values between xlo and xhi, ([xlo,xhi,N,r]) gives randomly distributed values (each trial of the simulation correspond to a different realization), ([xlo,xhi,N,l]) gives log distributed values.
- Numerical values without dots or "e" or "E" are assumed to be integers, unless they are in square brackets, where they are assumed to be floats (except for random values, which can be integers).
- String parameters bounded by underscores (e.g. _a_vector_) are assumed to be names of REAL4 sequences passed by the datacondAPI.
- The beginning of the data segment defines the origine of time at the center of the Earth. argv[9] gives the delay between that time and the beginning of the signal time-series, in number of samples, and at the center of the Earth. Given the source position and the interferometer, an additional delay for the wave propagation is added.
- argv[4-7] do not accept arguments of the form ([xlo,xhi,N,r]) with N¿1. When any of these arguments is random and more than one waveform is injected per segment, the same realization of the random argument is used for all injected waveforms. If a list is passed, the list must have the same length for all argv[4-7], and the values of the various arguments will be 'locked' together, i.e. will be used as a single table.
- If argv[5] or argv[6] are outside the allowed range, the waveform is injected with $F_+ = F_\times = 1$ and no time delay with respect to the earth center.
- argv[4-7] and argv[9] can be matrices in order to perform coherent injections for many IFOs. They are passed as ilwd files of dimension (number of segments, injections per segment), using double underscores (e.g. __file__). argv[8] must then be equal to the product of the dimensions of the ilwd file.
- The GW data must be passed with name containing GW_STRAIN_DATA .
- The code uses LALExtractResponse to get the response function.
- The first argument can be set to binOutput while `argv[i] -> argv[i+1]` in order to save the output in a binary format. This leads to up to 2x improvements in speed under LDAS.
- The first or second argument can alternatively be set to fileOutput:path in order to bypass the ligolwAPI completely and speed up large jobs.
- The first or second argument can be set to noLALBurstOutput in order to bypass the burst parameter estimation function. The unaltered output of the ETG is then reported.
- If argv[9] is a list of injection times (esp. random), only those injections for which the signals do not overlap will be performed and reported. The check for overlapping signals is not performed for matrix inputs.
- The first argument can be set to fileOutput:file in order to bypass the LIGOLWAPI and dump data in binary format to file.

Chapter 19

Shared object stochastic

Implements a stochastic background search using the standard optimally-filtered cross-correlation statistic. This shared object can now handle data from both interferometers and resonant bar detectors, and also either heterodyned or non-heterodyned data streams.

Note: Since no specific mechanism exists for reading the units string from an ILWD into the LALUnit structure, this DSO assumes its inputs have the expected units. These are

- Data Streams $h^{\mathrm{W}}_{1,2}(t)$: ADC counts
- PSDs $P^{\mathrm{W}}_{1,2}(f)$: (ADC counts)^2 / Hz
- Response Functions $\tilde{R}_{1,2}(f)$: ADC counts / attostrain $= 10^{18}$ counts / strain

These are set "by hand" in LALConditionData() without looking at the relevant units fields in the multiDimData. This should be changed to parse the units fields and construct the appropriate LALUnit structures, or at least check to see if it matches one of a set of predefined strings.

Note also that the frequency domain response function expected by the LAL stochastic package routines converts gravitational wave strain into detector output, not the other way around.

19.1 Filter parameters

```
# libldasstochastic.so should be invoked with the following arguments
#
# argv[1]  = numSegments       Number of data segments
# argv[2]  = numPoints         Number of points in a data segment
# argv[3]  = baseFreq          Hetrodyning base frequency of time series data
# argv[4]  = windowType1       Index of window type (0=rect, 1=Hann, etc)
# argv[5]  = windowTop1        Width of flat top in middle of window
# argv[6]  = windowType2       Index of window type (0=rect, 1=Hann, etc)
# argv[7]  = windowTop2        Width of flat top in middle of window
# argv[8]  = fullLength        Length after zero-padding
# argv[9]  = numFrequencies    Number of frequencies in optimal filter
# argv[10] = f0                Start frequency for optimal filter
# argv[11] = deltaF            Frequency spacing for optimal filter
# argv[12] = outputNumFreq     Number of frequencies in CC spectrum
# argv[13] = outputF0          Start frequency for CC spectrum
# argv[14] = outputDeltaF      Frequency spacing for CC spectrum
# argv[15] = epochsMatch       Check that time streams are simultaneous
# argv[16] = siteID1           Identifier for first detector geometry
# argv[17] = siteID2           Identifier for second detector geometry
# argv[18] = outputType        Bitmask for DSO outputs
#
```

19.2 Notes

- See the LAL stochastic package for details about stochastic searches using the standard optimally-filtered cross-correlation statistic.
- If `baseFreq` is zero, the shared object expects the input time series to be real, while if it's non-zero, they are assumed to be complex heterodyned time series. Neither the base frequency nor the inittial heterodyning phase is read in from the ILWD input to the wrapper.
- The recognized site identifiers are:

i	Interferometer (differential mode), geometry data specified in input to wrapper
b	Cylindrical bar detector, geometry data specified in input to wrapper
H	LIGO Hanford
L	LIGO Livingston
G	GEO-600
A	ALLEGRO (IGEC orientation)
A###	ALLEGRO with an orientation of ### degrees East (*clockwise*) of North
integer (0-9)	The detector in `lalCachedDetectors[]` with the corresponding index; this is added for backwards compatibility only; the one-letter identifiers are preferred if the geometry has to be specified in the filter parameters and not read from frames.

  The `i` and `b` options should be used under normal circumstances.
- The window type is taken from the `WindowType` enum defined in the `window` package of LAL. It optionally has a flat (windowing function = 1) top of width `windowTop` inserted in the middle. The net effect is, if numPoints $= N$, fullLength $= M$, and windowTop $= P$, to construct a window $\{w_j | j = 0, \ldots, N-P-1\}$ of length $N-P$ and make a zero-padded, windowed data stream according to

$$\bar{h}[k] = \begin{cases} w_k\, h[k] & k = 0, \ldots, \left\lceil \frac{N-P-1}{2} \right\rceil \\ h[k] & k = \left\lceil \frac{N-P-1}{2} \right\rceil + 1, \ldots, \left\lceil \frac{N-P-1}{2} \right\rceil + P \\ w_{P+k}\, h[k] & k = \left\lceil \frac{N-P-1}{2} \right\rceil + P + 1, \ldots, N-1 \\ 0 & k = N, \ldots, M-1 \end{cases} \tag{19.1}$$

- `fullLength` should be at least $2N-1$, although it may be convenient to use $2N$ instead.
- The `outputType` field is a decimal number which describes which combination of outputs from the DSO is desired. The options are

1	Cross-correlation statistic, written to the metadata database
2	Theoretical variance of the cross-correlation statistic, written to the metadata database
4	Cross-correlation spectrum (integrand of the cross-correlation statistic), written to the metadata database
8	Cross-correlation spectrum (integrand of the cross-correlation statistic), written to proc frames

  The options can be combined, so that for instance `outputType=10` means to write the cross-correlation spectrum to frames and the theoretical variance to the database.

19.3 Header Stochastic.h

Provides structures and definitions global to the stochastic search, notably StochasticSearchParams.

Synopsis

```
#include "Stochastic.h"
```

Error conditions

There are no error conditions associated with this header file.

Structures and Unions

Union StochasticFFTPlan

This holds a pointer to an FFTPlan, with the type being determined by whether or not the time series data for the search are heterodyned. The fields are:

```
RealFFTPlan *realPlan
```

```
ComplexFFTPlan *complexPlan
```

Structure StochasticSearchParams

The fields are:

```
BOOLEAN searchMaster
```

```
BOOLEAN haveData
```

```
UINT4 currentSegment
```

```
COMPLEX8FrequencySeries *optimalFilter
```

```
REAL4WithUnits *sigma
```

```
REAL4Vector *window1
```

```
REAL4Vector *window2
```

```
StochasticFFTPlan fftPlan
```

```
StochasticInitParams *initParams
```

```
StochasticDataSegmentVector *stochasticSegVec
```

```
gpsTimeInterval timeInterval
```

```
REAL4 segmentDuration
```

19.4 Header StochasticData.h

Provides structures needed for StochasticSearchParams, notably including StochasticInitParams, StochasticDataSegment, and StochasticDataSegmentVector.

Synopsis

```
#include "StochasticData.h"
```

Error conditions

<*name*>	code	description
NNUL	1	"Non-null pointer"
NULL	2	"Null pointer"
NSEG	3	"Number of data segments is zero or negative"
NPTS	4	"Number of points in a data segment is zero or negative"
NFRQ	5	"Number of frequencies is zero or negative"
ALOC	6	"Memory allocation error"
STID	7	"Detector Site ID out of range"
UEXT	8	"Unrecognised exchange type"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBData.h on line 1.60.

Structures and Unions

Structure StochasticInitParams

The fields are:

UINT4 numSegments Number of data segments

UINT4 numPoints Number of points in a data segment

REAL8 baseFreq Hetrodyning base frequency of time series data

WindowType windowType1 Type of window to be used on first data stream

UINT4 windowTop1 Width of flat top for first window

WindowType windowType2 Type of window to be used on second data stream

UINT4 windowTop2 Width of flat top for second window

UINT4 fullLength Length after zero-padding

UINT4 numFrequencies Number of frequencies in optimal filter

REAL8 f0 Start frequency for optimal filter

REAL8 deltaF Frequency spacing for optimal filter

UINT4 outputNumFreq Number of frequencies in cross-correlation spectrum

REAL8 outputF0 Start frequency for CC spectrum

REAL8 outputDeltaF Frequency spacing for CC spectrum

BOOLEAN epochsMatch Check that time streams are simultaneous

CHAR siteID1 Identifier for first detector

CHAR siteID2 Identifier for second detector

REAL4 *azimuth1 Azimuth of first detector in degrees East of North (null if not applicable)

REAL4 *azimuth2 Azimuth of second detector in degrees East of North (null if not applicable)

UINT4 outputType Bitmask for DSO outputs

Structure StochasticRealDataSegment

The fields are:

REAL4TimeSeries *data1

REAL4TimeSeries *data2

Structure StochasticComplexDataSegment

The fields are:

COMPLEX8TimeSeries *data1

COMPLEX8TimeSeries *data2

Structure StochasticRealDataSegmentVector

The fields are:

UINT4 length

StochasticRealDataSegment *data

Structure StochasticComplexDataSegmentVector

The fields are:

UINT4 length

StochasticComplexDataSegment *data

Union StochasticDataSegmentVector

The fields are:

StochasticRealDataSegmentVector *realVec

StochasticComplexDataSegmentVector *complexVec

19.4.1 Module StochasticData.c

Routines for creating and destroying a StochasticRealDataSegmentVector or StochasticComplexDataSegmentVector

Prototypes

```
void LALSCreateStochasticDataSegmentVector (                    1.50
    LALStatus*                      status,                     SGWBData.c
    StochasticRealDataSegmentVector**   vector,
    StochasticInitParams*           params
)

void LALSDestroyStochasticDataSegmentVector(                    1.187
    LALStatus*                      status,                     SGWBData.c
    StochasticRealDataSegmentVector**   vector
)

void LALCCreateStochasticDataSegmentVector (                    1.252
    LALStatus*                      status,                     SGWBData.c
    StochasticComplexDataSegmentVector**   vector,
    StochasticInitParams*           params
)

void LALCDestroyStochasticDataSegmentVector(                    1.389
    LALStatus*                      status,                     SGWBData.c
    StochasticComplexDataSegmentVector**   vector
)
```

Description

Creates and destroys a StochasticRealDataSegmentVector or StochasticComplexDataSegmentVector, which holds the input time-series data for two interferometers.

Algorithms

Uses

- LALMalloc
- LALFree
- LALSCreateVector
- LALSDestroyVector
- LALCCreateVector
- LALCDestroyVector

Notes

```
Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.          1.1
$Id: SGWBData.c,v 1.2 2003/09/29 20:11:29 kaice Exp $           SGWBData.c
```

19.5 Header StochasticInitSearch.h

Header file for a routine that checks the validity of the input parameters, initializes them, and then allocates memory for certain data structures used in the stochastic search.

Synopsis

```
#include "StochasticInitSearch.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
ALOC	3	"Memory allocation error"
ARGS	4	"Wrong number of arguments"
ARG0	5	"Wrong value for argv[0]"
NSEG	6	"Number of data segments is zero or negative"
NODENSEG	7	"Number of segments per nodeis zero or negative"
NPTS	8	"Number of points in a data segment is zero or negative"
BASE	9	"Heterodyning base frequency is negative"
UWIN	10	"Unknown window type"
NWIN	11	"Flat top longer than window"
NFRQ	12	"Number of frequencies is zero or negative"
F0	13	"Start frequency is negative"
DELF	14	"Frequency spacing is zero or negative"
DTSZ	15	"Nonstandard size of COMPLEX8 and/or UCHAR"
FRES	16	"Frequency sampling for output finer than for optimal filter"
FRNG	17	"Frequency range for output not fully contained in optimal filter"
FNUM	18	"Number of frequencies in optimal filter is greater than number of data elements"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBInitSearch.h on line 1.80.

Structures

None.

19.5.1 Module StochasticInitSearch.c

Routine that checks the validity of the input parameters, initializes them, and then allocates memory for certain data structures used in the stochastic search.

Prototypes

1.56
SGWBInitSearch.c

```
void LALInitSearch(
    LALStatus*              status,
    void**                  searchParams,
    LALInitSearchParams*    initSearchParams
)
```

Description

Checks the validity of the input parameters, initializes them, and then allocates memory for the time-series input data, optimal filter, and windowing functions.

Algorithms

Uses

- LALMalloc
- LALCCreateVector
- LALCreateForwardRealFFTPlan
- LALCreateStochasticDataSegmentVector

Notes

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBInitSearch.c,v 1.7 2003/10/07 17:17:29 jolien Exp $

19.6 Header StochasticConditionData.h

Header file for a routine that does one-time processing in preparation for the stochastic search.

Synopsis

```
#include "StochasticConditionData.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"
MSTR	2	"Not search master"
NSEQ	3	"Wrong number of sequences in InPut structure"
TYPE	4	"Input data has wrong type"
DIMS	5	"Input data has wrong dimensions"
SIZE	6	"Input data has wrong size"
NAME	7	"Wrong data name in InPut structure"
ALOC	8	"Memory allocation error"
UNIT	9	"Input data had unexpected units"
STID	10	"Detector site identifier unrecognized"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBConditionData.h on line 1.61.

Structures

None.

19.6.1 Module StochasticConditionData.c

Routine for performing one-time processing in preparation for the stochastic search.

Prototypes

1.131
SGWBConditionData.c

```
void LALConditionData (
                       LALStatus*          status,
                       LALSearchInput*     inout,
                       void*               searchParams,
                       LALMPIParams          *mpiParams
                       )
```

Description

Performs one-time processing (e.g., extraction of input data, construction of windowing functions, and calculation of the optimal filter for a stochastic background having $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$) in preparation for the stochastic search.

Algorithms

LALConditionData does the following:

1. Constructs the windowing functions requested for the two data streams
2. Checks the validity of the input data, and extracts the detector geometry, time-series data, power spectral densities, and instrument response functions from the multidim input data.
3. Converts 2-sided power spectral densities (produced by the datacondAPI) to 1-sided power spectra having the appropriate bandwidth and frequency resolution for construction of the optimal filter.
4. Coarse-grains the instrument response functions to the appropriate bandwidth and frequency resolution for construction of the optimal filter.
5. Casts the extracted input data to types expected by the LAL stochastic routines.
6. Calculates the overlap reduction function given the detector geometry information.
7. Sets $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$ for this particular search.
8. Calculates inverse calibrated and half-calibrated power spectra given the calibrated power spectra and instrument response functions.
9. Calculates the optimal filter for a stochastic background having $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$ using the previously calculated information.

Uses

- LALMalloc
- LALFree
- LALCreateVector
- LALCCreateVector
- LALSCreateVector
- LALDestroyVector
- LALCDestroyVector
- LALSDestroyVector
- LALSCoarseGrainFrequencySeries
- LALCCoarseGrainFrequencySeries
- LALStochasticOmegaGW
- LALStochasticInverseNoise
- LALStochasticOptimalFilter
- LALStochasticOptimalFilterNormalization
- CastToReal4
- CastToComplex8

Notes

- LALConditionData calls two static functions CastToReal4 and CastToComplex8, which are used to cast time-series data, power spectral densities, and instrument response functions to the proper type for the LAL stochastic routines.
- Since no specific mechanism exists for reading the units string from an ILWD into the LALUnit structure, this DSO assumes its inputs have the expected units. These are
  - Data Streams $h^{\mathrm{W}}_{1,2}(t)$: ADC counts
  - PSDs $P^{\mathrm{W}}_{1,2}(f)$: (ADC counts)^2 / Hz
  - Response functions $\tilde{R}_{1,2}(f)$: ADC counts / attostrain $= 10^{18}$ counts / strain

  These are set "by hand" in LALConditionData() without looking at the relevant units fields in the multiDimData. This should be changed to parse the units fields and construct the appropriate LALUnit structures, or at least check to see if it matches one of a set of predefined strings.

19.7 Header SGWBApplySearch.h

Header file for a routine that implements the stochastic background search using the standard optimally-filtered cross-correlation statistic.

Synopsis

```
#include "SGWBApplySearch.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"
NNUL	2	"Non-null pointer"
MSTR	3	"Search master not being used"
RANK	4	"Search master has incorrect rank"
ALOC	5	"Memory allocation error"
SIZE	104	"Invalid data length"
ROWS	120	"Invalid number of rows"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBApplySearch.h on line 1.54.

Structures

struct StochasticSearchOutput

The fields are:

REAL4WithUnits *ccStatReal

COMPLEX8WithUnits *ccStatComplex

REAL4WithUnits *sigma

COMPLEX8FrequencySeries *ccSpec

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBApplySearch.h,v 1.2 2003/09/03 02:37:39 kaice Exp $

19.7.1 Module StochasticApplySearch.c

Routine that implements the stochastic background search, calculating the values of the spectrum of the optimally-filtered cross-correlation statistic, building the output for subsequent insertion into the metadata database and frames.

Prototypes

1.101 SGWBApplySearch.c

```
void LALApplySearch(
                    LALStatus             *status,
                    LALSearchOutput       *output,
                    LALSearchInput        *input,
                    LALApplySearchParams  *applyParams
                    )
```

Description

Implements the search for a stochastic background, calculating the values of the spectrum of the optimally-filtered cross-correlation statistic with $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$. The cross-correlation statistic spectrum values are then built for subsequent insertion into the metadata database and frames using the StochasticBuildOutput routine.

Algorithms

Given time-series data from two interferometers (pre-processed by the datacondAPI) and values of the optimal filter (calculated in LALConditionData), LALApplySearch does the following:

1. Splits the input time-series data into numSegments≥ 10 segments, each of length numPoints.
2. Windows, zero-pads, and FFTs each segment of time-series data.
3. Calculates the values of the standard optimally-filtered cross-correlation statistic for each segment of data, using the values of the optimal filter previously calculated in LALConditionData.
4. Coarse-grains the cross-correlation statistic spectrum output to the desired bandwidth and frequency resolution.
5. Builds the cross-correlation statistic spectrum output (for each data segment) for subsequent insertion into the metadata database and frames.

Uses

- LALCCreateVector
- LALSZeroPadAndFFT
- LALStochasticCrossCorrelationSpectrum
- LALCCoarseGrainFrequencySeries
- StochasticBuildOutput
- LALCDestroyVector

Notes

The analysis is run on *only* the search master node of the beowulf, since we are currently considering the simple case of stochastic signals with $\Omega_{\rm gw}(f) = \Omega_0 = \text{const}$. In the future, searches for stochastic backgrounds having different spectra (e.g., power law $\Omega_{\rm gw}(f) \propto f^{\alpha}$) should be run on multiple slaves (e.g., one node for each α).

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBApplySearch.c,v 1.7 2003/10/07 17:17:09 jolien Exp $

1.1 SGWBApplySearch.c

19.7.2 Module StochasticBuildOutput.c

Routine that builds the output of the optimally-filtered cross-correlation statistic spectrum for insertion into the metadata database and frames.

Prototypes

1.70 SGWBBuildOutput.c

```
void StochasticBuildOutput (
      LALStatus*                 status,
      LALSearchOutput*           output,
      StochasticSearchOutput*    input,
      StochasticSearchParams    *params
)
```

1.637 SGWBBuildOutput.c

```
void
LALCreateDBEntryBLOBU(
    LALStatus *status,
    dataBase  *output,
    LALBLOB   *blob,
    UINT4      rows
    )
```

Description

StochasticBuildOutput() builds the output of the optimally-filtered cross-correlation statistic spectrum for subsequent insertion into the metadata database (as a summ_spectrum table) and frames (as a sequence in filter_output).

LALCreateSearchSummvarsDatabase() creates a search_summvars database entry, analogous to LALCreateSearchSummaryDatabase(), while is a version of which treats the BLOB as (unsigned char *) rather than (signed char *).

Algorithms

Uses

- LALCreateSearchSummaryDataBase
- LALCreateSearchSummvarsDataBase
- LALCHARCreateVector
- UnitAsString
- LALCHARDestroyVector
- LALCreateNextDBEntry
- LALCreateDBEntryData
- LALCreateDBEntryBLOB
- LALCreateDBEntryBLOBU
- LALMalloc
- LALCalloc

Notes

1.1 SGWBBuildOutput.c

```
Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBBuildOutput.c,v 1.3 2003/09/03 02:37:17 kaice Exp $
```

19.8 Header StochasticFinalizeSearch.h

Header file for a routine that cleans up after the stochastic search.

Synopsis

```
#include "StochasticFinalizeSearch.h"
```

Error conditions

<*name*>	code	description
NULL	1	"Null pointer"

The status codes in the table above did not obey the LAL naming convention, i.e. the code does not use the file name in all caps as the prefix for the error names. Consult the source code for the full names. Better yet, fix the code. The source code with these messages is in SGWBFinalizeSearch.h on line l.46.

Structures

None.

19.8.1 Module StochasticFinalizeSearch.c

Routine that cleans up after the stochastic search.

Prototypes

1.51
SGWBFinalizeSearch.c

```
void LALFinalizeSearch(
    LALStatus*          status,
    void**              searchParams
)
```

Description

Cleans up after the stochastic search–i.e., frees memory previously allocated in the InitSearch routine.

Algorithms

Uses

- LALSDestroyStochasticDataSegmentVector
- LALCDestroyStochasticDataSegmentVector
- LALCDestroyVector
- LALFree
- LALDestroyRealFFTPlan
- LALDestroyComplexFFTPlan

Notes

Author: Anderson, W.G., Romano, J.D., and Whelan, J.T.
$Id: SGWBFinalizeSearch.c,v 1.4 2003/09/03 02:37:17 kaice Exp $

1.1
SGWBFinalizeSearch.c

Chapter 20

Package: stackslide

20.1 Introduction

20.1.1 Overview

The stackslide dynamic shared object (DSO) was written to search for continuous gravitational waves based on the algorithm developed in Brady and Creighton [1]. It is a contribution to LALwrapper and runs from within the wrapperAPI in stand-alone mode or the LDAS environment. The code can be checked out from the CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/lalwrapper and is located in the contrib/stackslide/ directories. The code uses functions from the LSC Algorithms Library (LAL) to process the input data. See the LAL Software Documentation, Chapters 29-30, available from http://www.lsc-group.phys.uwm.edu/lal/lsd.pdf. The LAL code can be checked out from the CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/lal and is located in the lal/packages/pulsar/ directories.

Scripts written in tcl that drive the stackslide DSO are in CVS repository: :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/lal under the dsorun/contrib/stackslide directory. These scripts send dataPipeline commands to the LDAS managerAPI to run the stackslide DSO.

The input to the code is frequency-domain data, either from SFTs, PSD's, or the $\mathcal{F}$-statistic, for a narrow band of frequencies. The total observation time that the analysis is applied to is divided into segments; we will call the frequency domain data from each time segment one Block of data (BLKs). The Blocks are combined coherently to produce Stacks (STKs). For each point in a given patch of the parameter space the STKs are slid to adjust for frequency evolution between STKs and then added incoherently to produce Sums (SUMs). Power in each frequency bin of the SUMs are checked against thresholds and are recorded in the LDAS database. SUMs are output into files if requested.

20.1.2 The Stack-Slide Algorithm

In the simplest case the STKs will be the power spectral densities computed from BLKs which are SFTs. The SFTs are just Discrete Fourier Transforms (DFTs) of the segments of the data:

$$\tilde{x}_k = \sum_{j=0}^{n-1} x_j e^{-2\pi i \frac{jk}{N}}. \tag{20.1}$$

Dimensionless STKs that are Power Spectral Densities (PSDs) (really periodograms) made from SFTs typically would be given by:

$$p_k = \left(\frac{(\Delta t)^2 \tilde{x}_k^* \tilde{x}_k / T_{\mathrm{SFT}}}{S_h} \right), \tag{20.2}$$

where S_h is the suitably averaged power spectral density of the noise for a narrow band that contains frequency bin k.

We expect in the optimal case the STKs will be the $\mathcal{F}$-statistic computed from BLKs which are SFTs. Dimensionless STKs that are the $\mathcal{F}$-statistic made from SFTs typically would be computed using LALDemod. (See the LAL documentation for LALDemod.)

For each parameters space point specified by sky position and spin evolution parameters, the STKs are slid to put the power for each initial frequency into the same bin of each STK. The frequency models are given by [1]:

$$f(t;\lambda) = f_0 \left(1 + \frac{\vec{v}}{c} \cdot \vec{n}\right) \left[1 + \sum_{s=1} f_s \left(1 + \frac{\vec{x}}{c} \cdot \vec{n}\right)^s\right], \tag{20.3}$$

where f_0 is the initial gravitational wave frequency at time $t = 0$, $\vec{x}$ and $\vec{v}$ are the position and velocity of the detector measured from the Solar System Barycenter (SSB) at the time the wave arrives at the detector, $\vec{n}$ is the unit vector that points from the SSB to the source, and the f_s are the spin evolution (usually spindown) parameters. Note that factors of $s!$ have been absorbed into the f_s (which is more computationally efficient when coding this equation). The slid STKs are summed to produce Sums (SUMs) and these are checked against thresholds for significance.

20.2 Header StackSlide.h

20.2.1 Structures

struct StackSlideSearchParams

This structure to hold all parameters and data that need to be initialized in LALInitSearch and LALConditionData and preserved between calls to LALApplySearch in LALWrapper.

```
typedef struct
tagStackSlideSearchParams
{

  /*****************************************/
  /*                                       */
  /* START SECTION: parameters passed as   */
  /* arguments.  Those indented are        */
  /* computed from these.                  */
  /*                                       */
  /*****************************************/

  UINT4   gpsStartTimeSec;           /* GPS start time of data requested seconds */
  UINT4   gpsStartTimeNan;           /* GPS start time of data requested nanoseconds */
  REAL8   sampleRate;                /* Sample rate of the time-domain data used to make the frequency-domain input data Blocks */
    REAL8   deltaT;                    /* Time step size in seconds = 1.0/sampleRate */
  REAL8   duration;                  /* Total time being analyzed  */
    UINT4   totalNumTimeSamples;       /* Total Number of time samples = sampleRate*duration */

  INT4    numBLKs;                   /* Number of input BLKs.  Not duration/tBLK if gaps are present */
  REAL8   tBLK;                      /* duration in seconds of BLKs */
    INT4    numBLKsPerDuration;        /* Equals duration/tBLK; but not acual number of blocks if gaps exist in the data */
    INT4    nSamplesPerBLK;            /* Equals sampleRate*tBLK = number of time domain data points used to compute one BLKs */
  REAL8   tEffBLK;                   /* Effective duration of BLK such that dfBLK = 1.0/tEffBLK  */
    REAL8   dfBLK;                     /* Freq resolution of BLK = 1.0/tEffBLK  Could be different from 1/tBLK due to oversampling
  REAL8   f0BLK;                     /* Start frequency of the input BLKs */
  REAL8   bandBLK;                   /* Band width of the input BLKs. */
  INT4    nBinsPerBLK;               /* Number of data points in the input BLK in the input frequency band */

  INT4    numBLKsPerSTK;             /* Number of BLKs to use to make one STK */
    INT4    numSTKs;                   /* Number of actual STKs. Not duration/tSTK if gaps are present */
    REAL8   tSTK;                      /* duration in seconds of STKs = tBLK*numBLKsPerSTK */
  REAL8   tEffSTK;                   /* Effective duration of STK such that dfSTK = 1.0/tEffSTK */
    REAL8   dfSTK;                     /* Freq resolution of STK = 1.0/tEffSTK. Could be different from 1/tSTK due to oversampling
  REAL8   f0STK;                     /* Start frequency of STKs */
  REAL8   bandSTK;                   /* Band width of the STKs. */
  INT4    nBinsPerSTK;               /* Number of data points in the STKs in the STK frequency band */

  INT4    numSTKsPerSUM;             /* Number of STKs to use to make one SUM (Usually duration/tSTK) */
    INT4    numSUMsPerParamSpacePt;  /* Number of output SUMs per parameter space point = params->duration/params->tSUM. (Usually wi
    REAL8   tSUM;                      /* duration in seconds of output SUMs = tSTK*numSTKsPerSUM. (Usually = duration) */
  REAL8   tEffSUM;                   /* Effective duration of SUM such that dfSUM = 1.0/tEffSUM */
    REAL8   dfSUM;                     /* Freq resolution of output SUMs.  dfSUM = 1.0/tEffSUM. Could be different from dfSTK in som
  REAL8   f0SUM;                     /* Start frequency of output SUMs */
  REAL8   bandSUM;                   /* Band width of the output SUMs. */
  INT4    nBinsPerSUM;               /* Number of data points in the output SUMs in the input frequency band */

  CHAR    *ifoNickName;              /* 2 character ifoNickName = H1, H2, or L1; note IFO is the site = LHO, LLO, GEO, etc.... */
  CHAR    *IFO;                      /* Identifies the interferometer site = LHO, LLO, GEO, etc.... */
  CHAR    *patchName;                /* Name of the patch in parameter space of the search (e.g, Galactic Center) */
  INT4    patchNumber;               /* Index of the patch in parameter space of the search (will be -1 until indexing implemented)

  CHAR    tDomainChannel[dbNameLimit];  /* Time domain channel name used to produce blocks */
  CHAR    fDomainChannel[dbNameLimit];  /* Frequency domain channel name that blocks are stored as */

  INT2    inputDataTypeFlag;         /* 0 = SFTs, 1 = PSDs, 2 = F-stat */

  INT2    parameterSpaceFlag;        /* 0 = use input delta for each param, 1 = params are input as vectors of data, 2 = use input p
    INT4    numParamSpacePts;          /* Total number of points in the parameter space to cover */
    INT4    numSUMsTotal;              /* Total Number of Sums output = numSUMsPerParamPt*numParamSpacePts */
```

```
INT2    stackTypeFlag;          /* 0 means stacks are PSDs from SFTs, 1 means stacks are F-statistic from SFTs */
INT4    Dterms;                 /* = terms used in the computation of the dirichlet kernel in LALDemod (Default value is 32) */

INT2    thresholdFlag;          /* How to apply the threshold; 0 = if power above threshold 1 put result in database */
REAL8   threshold1;             /* Threshold 1 for put results in the database (could be on power or false alarm rate) */
REAL8   threshold2;             /* Threshold 1 for put results in the database (could be on power or false dismisal rate) */

INT2    calibrationFlag;        /* Flag that specifies what calibration to do; -1 means Blks are already calibrated, 0 means le

INT2    weightFlag;             /* Flag that specifies whether to weight BLKs or STKs with a(t) or b(t).  */

INT2    windowFilterFlag;       /* Flag that specifies whether any filtering or windowing was done or should be done. (If < 0 t
REAL8   windowFilterParam1;     /* 1st parameter to use in windowing or filtering */
REAL8   windowFilterParam2;     /* 2nd paramter to use in windowing or filtering */
REAL8   windowFilterParam3;     /* 3rd paramter to use in windowing or filtering */

INT2    normalizationFlag;      /* Flag that specifies what normalization to do.  If < 0 then specifies what normalization was
REAL8   f0NRM;                  /* Start frequency to normalize over to create NRMs */
REAL8   bandNRM;                /* Band width to normalize over to create NRMs. */
INT4    nBinsPerNRM;            /* Number of data points to normalize over to create to creat NRMs */

INT2    testFlag;               /* Specifies any tests or debugging to do; 0 means no test */

UINT4   numSUMsPerCall;         /* If > 0 then = number of SUMs (parameter space points) to compute during each call to ApplySe
INT2    outputFlag;             /* Flag that specifies what to output; if > 0 then will output Sums into frame file */

INT2    unusedFlag1;            /* Place holder for future flag */
REAL8   unusedParam1;           /* Place holder for future parameter */
INT2    unusedFlag2;            /* Place holder for future flag */
REAL8   unusedParam2;           /* Place holder for future parameter */

REAL8   startRA;                /* Starting Right Ascension in radians */
REAL8   stopRA;                 /* Ending Right Ascension in radians */
REAL8   deltaRA;                /* Right Ascension step size in radians */
INT4    numRA;                  /* Number of RA to compute SUMs for */

REAL8   startDec;               /* Starting Declination in radians */
REAL8   stopDec;                 /* Ending Declination in radians  */
REAL8   deltaDec;                /* Declination step size in radians */
INT4    numDec;                  /* Number of Dec to compute SUMs for */

INT4    numSpinDown;            /* Number of nonzero spindown parameters */

REAL8   startFDeriv1;           /* Starting 1st deriv of freq in Hz/s */
REAL8   stopFDeriv1;            /* Ending 1st deriv of freq in Hz/s */
REAL8   deltaFDeriv1;           /* 1st deriv of freq step size in Hz/s */
INT4    numFDeriv1;             /* Number of 1st derivs to compute SUMs for */

REAL8   startFDeriv2;           /* Starting 2nd deriv of freq in Hz/s^2 */
REAL8   stopFDeriv2;            /* Ending 2nd deriv of freq in Hz/s^2 */
REAL8   deltaFDeriv2;           /* 2nd deriv of freq step size in Hz/s^2 */
INT4    numFDeriv2;             /* Number of 2nd derivs to compute SUMs for */

REAL8   startFDeriv3;           /* Starting 3rd deriv of freq in Hz/s^3 */
REAL8   stopFDeriv3;            /* Ending 3rd deriv of freq in Hz/s^3 */
REAL8   deltaFDeriv3;           /* 3rd deriv of freq step size in Hz/s^3 */
INT4    numFDeriv3;             /* Number of 3rd derivs to compute SUMs for */

REAL8   startFDeriv4;           /* Starting 4th deriv of freq in Hz/s^4 */
REAL8   stopFDeriv4;            /* Ending 4th deriv of freq in Hz/s^4 */
REAL8   deltaFDeriv4;           /* 4th deriv of freq step size in Hz/s^4 */
INT4    numFDeriv4;             /* Number of 4th derivs to compute SUMs for */

REAL8   startFDeriv5;           /* Starting 5th deriv of freq in Hz/s^5 */
REAL8   stopFDeriv5;            /* Ending 5th deriv of freq in Hz/s^5 */
REAL8   deltaFDeriv5;           /* 5th deriv of freq step size in Hz/s^5 */
INT4    numFDeriv5;             /* Number of 5th derivs to compute SUMs for */
```

```
  /*****************************************/
  /*                                       */
  /* END SECTION: parameters passed as     */
  /* arguments.  Those indented are        */
  /* computed from these.                  */
  /*                                       */
  /*****************************************/

  /*****************************************/
  /*                                       */
  /* START SECTION: other parameters       */
  /*                                       */
  /*****************************************/

  CHAR  *dsoName;         /* Name of this DSO */ /* 11/05/01 gam */

  /* Basic beowulf node descriptors */
  BOOLEAN searchMaster;           /* TRUE on the search master */
  UINT4   rank;                   /* Rank of this slave */
  UINT4   numSlaves;              /* Total number of slaves */
  UINT4   curSlaves;              /* Current number of slaves still working */
  UINT4   numNodes;               /*  07/12/02 gam  Should equal number of slaves + 1 for search master */

  UINT4  numDBOutput;             /* Used to keep track of the number of rows written to the database */

  gpsTimeInterval times;          /* The GPS start and end time of the data */

  LALUnit     unitBLK;            /* Unit BLKs are stored in (e.g., strain per root Hz) */

  FFT **BLKData;                   /* Container for BLKs */
  REAL4FrequencySeries **STKData;  /* Container for STKs */
  REAL4FrequencySeries **SUMData;  /* Container for SUMs */

  INT4 iMinBLK;      /* Index of minimum frequency in BLK band */
  INT4 iMaxBLK;      /* Index of maximum frequency in BLK band */
  INT4 iMinNRM;      /* Index of mimimum frequency to include when normalizing BLKs */
  INT4 iMaxNRM;      /* Index of maximum frequency to include when normalizing BLKs */

  LIGOTimeGPS *timeStamps;          /* Container for time stamps giving initial time for which each BLK was computed. */
  gpsTimeInterval *timeIntervals;   /* Array of time intervals; needed for building frame output */

  EphemerisData *edat; /* 07/10/02 gam Add EphemerisData *edat = pointer to ephemeris data to StackSlideSearchParams */

  UINT4 firstK;           /* Index used to keep track of which BLK, STK, or SUM to work on */
  UINT4 lastK;            /* Index used to keep track of which BLK, STK, or SUM to work on */
  BOOLEAN finishedBLKs;   /* Set equal to true when all BLKS for this job have been found in input data */
  BOOLEAN finishedSTKs;   /* Set equal to true when all STKs for this job have been created */
  BOOLEAN finishedSUMs;   /* Set equal to true when all BLKS for this job have been created */

  INT4 whichSTK;          /* which STK does BLK go with  */
  INT4 lastWhichSTK;      /* Last value of whichSTK does BLK go with  */

  REAL8 **skyPosData;     /* Container for Parameter Space Data */
  REAL8 **freqDerivData;  /* Container for Frequency Derivative Data */
  INT4 numSkyPosTotal;    /* Total number of Sky positions to cover */
  INT4 numFreqDerivTotal; /* Total number of Frequency evolution models to cover */

  FreqSeriesPowerStats *SUMStats; /* Container for statistics about each SUM */

  /*****************************************/
  /*                                       */
  /* END SECTION: other parameters         */
  /*                                       */
  /*****************************************/


}
```

```
StackSlideSearchParams;
```

struct FreqSeriesPowerStats

This structure to hold statistics about a power series.

```
typedef struct
tagFreqSeriesPowerStats
{
REAL8 pwMax;      /* max power */
REAL8 freqPWMax;  /* frequency associated with pwMax */
REAL8 pwMean;     /* mean power */
REAL8 pwStdDev;   /* std dev of power */
}
FreqSeriesPowerStats;
```

20.2.2 Error Codes

<*name*>	code	description
NULL	1	"Null pointer"
GPSTINT	2	"Unexpected GPS time interval"
DELTAT	3	"Invalid deltaT"
INPUTN	4	"Unexpected number of input data points"
TBLK	5	"Invalid tBLK"
RA	6	"Invalid right ascension"
DEC	7	"Invalid declination"
FREQ	8	"Invalid frequency +/- 0.5*band (could be negative, outside LIGO band, or too high for sample rate)"
FREQDERIV	9	"One of the frequency derivatives is possibly too large; frequency will evolve outside allowed band"
ALOC	10	"Memory allocation error"
NODATA	11	"No input data was found"
NDATA	12	"Invalid number of input data points"
TIMESTEP	13	"Incorrect input data time step"
STARTTIME	14	"Incorrect input data start time"
STOPTIME	15	"Incorrect input data stop time"
NBLK	16	"Invalid nSamplesPerBLK"
NTBLK	17	"nSamplesPerBLK and tBLK are inconsistent with deltaT"
MCOHBLKPERCALL	18	"Invalid numSUMsPerCall"
IFONICKNAME	19	"Invalid or null ifoNickName"
IFO	20	"Invalid or null IFO"
TARGETNAME	21	"Invalid or null Target Name"
CHANNEL	22	"Invalid or null tDomainChannel"
ISTARTTIME	23	"Requested GPS start time resulted in invalid index to input data."
MISSINGBLKDATA	24	"Some requested input BLK data is missing"
STARTFREQ	25	"Input BLK start frequency does not agree with that requested"
FREQSTEPSIZE	26	"Input BLK frequency step size does not agree with that expected"

The status codes in the table above are stored in the constants STACKSLIDEH_E<*name*>, and the status descriptions in STACKSLIDEH_MSGE<*name*>. The source code with these messages is in StackSlide.h on line 1.85.

20.3 Module StackSlide.c

All the source code for the stackslide DSO currently resides in a single module: StackSlide.c.

20.3.1 Dependencies

The following include files are needed to build the stackslide DSO (some are already included in LALDemod.h):

```
#include <stdio.h>
#include <math.h>
#include <lal/LALStdlib.h>
#include <lal/AVFactories.h>
#include <ExtractSeries.h>
#include <lal/VectorOps.h>
#include <lal/RealFFT.h>
#include <lal/LALConstants.h>
#include <lal/LALDemod.h>
#include <lal/Units.h>
#include <StackSlide.h>
#include <lal/BandPassTimeSeries.h>
```

20.3.2 Static Functions

Besides the standard LALInitSearch, LALConditionData, LALApplySearch, and LALFinalizeSearch The StackSlide.c module also contains static functions for unpacking ephemeris data, handling time stamps for BLKs, and building database and multidim data output (output by LDAS in ilwd or frame format).

20.3.3 Preprocessor Flags

The following preprocessorflags can be uncommented/commented to turn on/off certain features of the code, mostly for debugging.

```
#define INCLUDE_BUILDSUMFRAME_CODE
/* #define DEBUG_STACKSLIDE */
/* #define DEBUG_PARAMETERS */
/* #define DEBUG_BLK_CODE */
/* #define DEBUG_DEMOD_CODE */
/* #define DEBUG_BUILDOUTPUT_CODE */
/* #define DEBUG_BUILDSUMFRAMEOUTPUT_CODE */
/* #define DEBUG_FRACREMAINING_CODE */
/* #define DEBUG_INPUTBLK_CODE */
/* #define DEBUG_LALWRAPPERINITBARYCENTER */
/* #define DEBUG_NORMALIZEBLKS */
/* #define DEBUG_CALIBRATEBLKS */
/* UNCOMMENT ONLY ONE OF INCLUDE_LALINITBARYCENTER OR INCLUDE_LALWRAPPERINITBARYCENTER AT A TIME */
/* UNCOMMENT INCLUDE_LALINITBARYCENTER FOR TEST PURPOSES IN STANDALONE MODE; OTHERWISE UNCOMMENT INCLUDE_LALWRAPPERINITBARYCENTER */
/* NOTE: when INCLUDE_LALINITBARYCENTER is uncommeted one must hard code input earth and sun file names (see below). */
/* #define INCLUDE_LALINITBARYCENTER */
#define INCLUDE_LALWRAPPERINITBARYCENTER
```

20.3.4 LALWRAPPERInitBarycenter.c

The function in LALWRAPPERInitBarycenter.c transfers ephemeris data from LDAS structures into LAL structures. It is necessary to use this function when inputing ephemeris data from LDAS using the -responsefiles option, for example like this:

```
-responsefiles { file:/ldas_outgoing/jobs/responsefiles/earth01.ilwd,pass
file:/ldas_outgoing/jobs/responsefiles/sun01.ilwd,pass }
```

Note that for now that this function is hard coded into StackSlide.c.

In stand-alone mode it is possible to input ephemeris data from LAL using the LALInitBarycenter function, by uncommenting the INCLUDE_LALINITBARYCENTER flag and commenting the INCLUDE_LALWRAPPERINITBARYCENTER flag.

20.4 Filter Parameters

Below are the parameters that must be defined by the user and passed through to the stackslide DSO via the -filterparams option. This option is included in a dataPipeline command that is sent to LDAS, or in a schema file used with the wrapperAPI in stand-alone mode. All parameters must be given, even if unused, in the given order, except unused parameters can be left off at the end of the list.

20.4.1 Time Domain Parameters

```
UINT4   gpsStartTimeSec
UINT4   gpsStartTimeNan
REAL8   sampleRate
REAL8   duration
```

gpsStartTimeSec = the whole number of seconds part of the GPS start time of the data that is being analyzed.

gpsStartTimeNan = the nanoseconds part of the GPS start time of the data that is being analyzed. (Usually will be 0.)

sampleRate = sample rate in Hz of the time-domain data used to make the frequency-domain input data Blocks. This gives the time step size deltaT = 1.0/sampleRate.

duration = the duration of the data that is being analyzed. Thus, if the data is analysized for the interval [start_time end_time) then end_time = start_time + duration.

20.4.2 Block Parameters

```
INT4    numBLKs
REAL8   tBLK
REAL8   tEffBLK
REAL8   f0BLK
REAL8   bandBLK
INT4    nBinsPerBLK
```

numBLKs = the actual number of input Blocks of data. This is not the total duration divided by the duration of one Block if gaps are present.

tBLK = duration in seconds of one Block of data.

tEffBLK = the effective duration of a Block such that the frequency step size for a Block = 1.0/tEffBLK.

f0BLK = the start frequency of the Blocks.

bandBLK = the bandwidth of the Blocks.

nBinsPerBLK = number of frequency bins in each BLOCK. Note that this probably should alway equal bandBLK*tEffBLK. Thus if start_freq = f0BLK and end_freq = f0BLK + bandBLK, then the frequencies included in the band are [start_freq end_freq). However, if nBinsPerBLK = bandBLK*tEffBLK + 1, then a convention of [start_freq end_freq] is being used to describe the frequency interval.

20.4.3 Stack Parameters

```
INT4    numBLKsPerSTK
INT4    tEffSTK
REAL8   f0STK
REAL8   f0STK
INT4    nBinsPerSTK
```

numBLKsPerSTK = number of consecutive Blocks to use to make one Stack. (The stack making algorithm will handle missing blocks when gaps exist in the data.)

tEffSTK = the effective duration of a Stack such that the frequency step size for a Block = 1.0/tEffSTK.

f0STK = start frequency of Stacks.

bandSTK = bandwidth of the Stacks.

nBinsPerSTK = number of frequency bins in one stack.

20.4.4 Sum Parameters

```
INT4    numSTKsPerSUM
REAL8   tEffSUM
REAL8   f0SUM
REAL8   bandSUM
INT4    nBinsPerSUM
```

numSTKsPerSUM = number of Stacks to use to make one Sum. (Usually all the Stacks will be used, but one could divide the analysis time into several stack-slide searches.)

tEffSUM = effective duration of a Sum such that dfSUM = 1.0/tEffSUM.

f0SUM = start frequency of the Sums.

bandSUM = bandwidth of the Sums.

nBinsPerSUM = number of frequency bins in one Sum.

20.4.5 IFO and Patch Parameters

```
CHAR    *ifoNickName
CHAR    *IFO
CHAR    *patchName
INT4    patchNumber
CHAR    tDomainChannel[dbNameLimit]
CHAR    fDomainChannel[dbNameLimit]
```

ifoNickName = the 2 character ifoNickName = H1, H2, or L1 of the interferometer the data came from.

IFO = identifies the interferometer site = LHO, LLO, GEO, etc....

patchName = the name of the patch in parameter space of the search (e.g, Galactic Center).

patchNumber = an index for the patch in parameter space being searched (will be unused until implemented).

tDomainChannel = time domain channel name used to produce Blocks. For example, this will usually be H2:LSC-AS_Q, H1:LSC-AS_Q, or L1:LSC-AS_Q.

fDomainChannel = frequency domain channel name used to request data from Blocks.

20.4.6 Flag Parameters

```
INT2    inputDataTypeFlag
```

inputDataTypeFlag = a flag that determines the type of the input data.

Case 0: Input data is SFTs.

Case 1: Input is PSDs.

Case 2: Input is F-statistic.

```
INT2    parameterSpaceFlag
```

parameterSpaceFlag = a flag that describes how to contruct the parameter space. Initially only case 0 will be implemented.

Case 0: use starting value, step size, and number of values input to generate a flat rectangular parameter space.

Case 1: use input vectors that contain the parameter space data.

Case 2: use input parameter space metric to generate the parameter space.

Case 3: Call LAL functions to generate the parameter space metric and use this to generate the parameter space.

```
INT2    stackTypeFlag
INT4    Dterms
```

stackTypeFlag = a flag that describes what type of Stacks to create.

Case 0: Stacks are PSDs created from Blocks.

Case 1: Stacks are the F-statistic made from SFTs. (Blocks must be SFTs in this case.)

Dterms = terms used in the computation of the dirichlet kernel in LALDemod (Default value is 32) when Stacks are the F-statistic.

```
INT2    thresholdFlag
REAL8   threshold1
REAL8   threshold2
```

thresholdFlag = a flag that describes what type of Stacks to create. Only one case will be initially supported. In the future different cases involving false alarm rates and false dismisal rates could be implemented.

Case 0: If Sum power is above threshold1 then put result into the database.

threshold1 = a threshold for putting results in the database (could be on cutoff on power or the false alarm rate).

threshold2 = a threshold for putting results in the database (could be another cutoff on power or false dismisal rate).

```
INT2    calibrationFlag;
```

calibrationFlag = a flag that describes what calibration is needed. Only Case -1 and Case 0 will be initially supported.

Case -1: Blocks are already calibrated. No further calibration is needed.

Case 0: Blocks are uncalibrated and no calibration is needed.

Case 1: Blocks are uncalibrated and calibration is needed.

```
INT2    weightFlag
```

weightFlag = a flag that describes whether to weight Blocks or Stacks with the beam pattern response. Note that this is different from normalization, which is described below. Only one case will be initially supported.

Case 0: No weighting is to be done.

```
INT2    windowFilterFlag
REAL8   windowFilterParam1
REAL8   windowFilterParam2
REAL8   windowFilterParam3
```

windowFilterFlag = a flag that describes any filtering or windowing that was done or should be done. (If ¡ 0 then specifies what was done in the time domain.)

Case 0: No windowing or filtering is to be done or was done.

windowFilterParam1 = 1st parameter to use in windowing or filtering.

windowFilterParam2 = 2nd paramter to use in windowing or filtering.

windowFilterParam3 = 3rd paramter to use in windowing or filtering.

```
INT2    normalizationFlag
REAL8   f0NRM
REAL8   bandNRM
INT4    nBinsPerNRM
```

normalizationFlag = a flag that describes any normalization that should be done. Only Cases 0 and 1 will be initially implemented.

Case -2: Blocks are already normalized using PSD estimated from running median using indicated band.

Case -1: Blocks are already normalized using PSD estimated from averages using indicated band.

Case 0: Blocks are unnormalized; no normalization needed.

Case 1: Normalize Blocks using PSD estimated from averages using indicated band.

Case 2: Normalize Blocks using PSD estimated from running median using indicated band.

f0NRM = Start frequency used to generated normalization.

bandNRM = bnand width used to generate normalization.

nBinsPerNRM = number of frequency bins to use to generate normalization.

```
INT2    testFlag
```

testFlag = a flag that specifies any tests or debugging to do.

Case 0: No test case or debugging is turned on.

```
UINT4   numSUMsPerCall
INT2    outputFlag
```

numSUMsPerCall = an LDAS specific parameter that specifies the number of SUMs to compute during each call to ApplySearch between reported progress to LDAS. (If ¡= 0 then this parameter will be ignored and all SUMs will be computed before reporting any progress to LDAS.)

outputFlag = a flag that specifies what to output. If file output is requested then LDAS will output Sums into frame or xml file. (Driver script call to LDAS will specify the output format.)

Case -1: Do not output any results; just output how many results would have been output (for test purposes).

Case 0: Output results to database only.

Case 1: Output results to database and output Sums to file.

```
INT2    unusedFlag1
REAL8   unusedParam1
INT2    unusedFlag2
REAL8   unusedParam2
```

Place holder for future flags and parameters. (Will preserve ordering of parameters for backwards compatibility.)

20.4.7 Sky Patch Parameters

```
REAL8   startRA
REAL8   stopRA
REAL8   deltaRA
INT4    numRA

REAL8   startDec
REAL8   stopDec
REAL8   deltaDec
INT4    numDec
```

startRA = Starting Right Ascension in radians.

stopRA = Ending Right Ascension in radians.

deltaRA = Right Ascension step size in radians.

numRA = Number of RA to compute SUMs for.

startDec = Starting Declination in radians.

stopDec = Ending Declination in radians.

deltaDec = Declination step size in radians.

numDec = Number of Dec to compute SUMs for.

There is redundancy here. However, start and stop values may be used to describe a patch on the sky, while delta and num values imply whether stop values are inclusive or exclusive.

WARNING: If parameterSpaceFlag is not zero these values may be ignored or used as bounds on a parameter space that is input or generated in another way depending on the value of this flag.

20.4.8 Spin Evolution Parameters

```
INT4    numSpinDown

REAL8   startFDeriv1
REAL8   stopFDeriv1
REAL8   deltaFDeriv1
INT4    numFDeriv1

REAL8   startFDeriv2
REAL8   stopFDeriv2
REAL8   deltaFDeriv2
INT4    numFDeriv2

REAL8   startFDeriv3
REAL8   stopFDeriv3
REAL8   deltaFDeriv3
INT4    numFDeriv3

REAL8   startFDeriv4
REAL8   stopFDeriv4
REAL8   deltaFDeriv4
INT4    numFDeriv4

REAL8   startFDeriv5
REAL8   stopFDeriv5
REAL8   deltaFDeriv5
INT4    numFDeriv5
```

numSpinDown = Number of nonzero spin evolution parameters that are specified. Currently this must be less than or equal to five. Only the nonzero spin evolution parameters need to be specified.

startFDeriv1 = Starting 1st deriv of freq in Hz/s.

stopFDeriv1 = Ending 1st deriv of freq in Hz/s.

deltaFDeriv1 = 1st deriv of freq step size in Hz/s.

numFDeriv1 = Number of 1st derivs to compute SUMs for.

startFDeriv2 = Starting 2nd deriv of freq in Hz/s$\hat{2}$.

stopFDeriv2 = Ending 2nd deriv of freq in Hz/s$\hat{2}$.

deltaFDeriv2 = 2nd deriv of freq step size in Hz/s$\hat{2}$.

numFDeriv2 = Number of 2nd derivs to compute SUMs for.

startFDeriv3 = Starting 3rd deriv of freq in Hz/s$\hat{3}$.

stopFDeriv3 = Ending 3rd deriv of freq in Hz/s$\hat{3}$.

deltaFDeriv3 = 3rd deriv of freq step size in Hz/s$\hat{3}$.

numFDeriv3 = Number of 3rd derivs to compute SUMs for.

startFDeriv4 = Starting 4th deriv of freq in Hz/s$\hat{4}$.

stopFDeriv4 = Ending 4th deriv of freq in Hz/s$\hat{4}$.

deltaFDeriv4 = 4th deriv of freq step size in Hz/s$\hat{4}$.

numFDeriv4 = Number of 4th derivs to compute SUMs for.

startFDeriv5 = Starting 5th deriv of freq in Hz/s$\hat{5}$.

stopFDeriv5 = Ending 5th deriv of freq in Hz/s$\hat{5}$.

deltaFDeriv5 = 5th deriv of freq step size in Hz/s$\hat{5}$.

numFDeriv5 = Number of 5th derivs to compute SUMs for.

There is redundancy here. However, start and stop values may be used to describe a patch on the sky, while delta and num values imply whether stop values are inclusive or exclusive.

WARNING: If parameterSpaceFlag is not zero these values may be ignored or used as bounds on a parameter space that is input or generated in another way depending on the value of this flag.

20.5 dataPipeline Commands

20.5.1 dataPipeline commands for Blocks = SFTs, Stacks = PSDs

Note that the example below needs to be updated.

```
cmd = { dataPipeline
-returnprotocol file:...
-subject { stack-slide job }
-mddapi frame
-database ldas_tst
-dynlib libldasstackslide.so.0.0.0
-np 2
-filterparams (...)
-datacondtarget ...
-aliases {...}
-algorithms {...}
-framequery {...}
}
```

20.5.2 dataPipeline commands for Blocks = SFTs, Stacks = F-statistics

...

20.6 Scripts

Scripts that drive the stackslide DSO and/or that perform auxilary actions will be in CVS repository :pserver:username@gravity.phys.uwm.edu:/usr/local/cvs/lalwrapper under the dsorun/contrib/stackslide/scripts directory.

20.6.1 Driver Scripts

This script can be used to run a single LDAS job, or loop over time through jobs. The script requires tcl and that the ligotools job package be installed. The job to be run is defined in a separate job file that is sourced. The job file must set the cmd variable which hold the command to send to LDAS.

Note that this needs to be updated.

```
DriveKPDJobs.tclsh

Syntax: ./DriveKPDJobs.tclsh [-v] [-s <site>] [-ft <frametype>]
 [-ifo <interferometer>] [-ch <channel>]
 [-db <database>] [-seg <seggroup>]
 [-segfile <segfile> -scol <num> -ecol <num> -qch <chname> -rmqc ]
 [-st <startTime>] [-dt <deltaT>] [-et <endTime>] [-e <emailaddress>]
 -log <logfile> [-alog] [-blog] [-rec <recordfile>] [-arec] [-brec]
 [-run] -job <jobfile>

-v        Print output to stdout.
-s        LDAS site to run job.  Allowed sites are lho, llo, mit, uwm, sw, dev, and test.
-ft       Frame type name to substitute for FRAME_TYPE in job file (e.g., R).
-ifo      Interferometer to substitute for INTERFEROMETER in job file (e.g., H, L, or LH).
-ch       Channel name to substitute for CHANNEL_NAME in job file (e.g., H2:LSC-AS_Q).
-db       LDAS database to use, e.g., lho_test, llo_test. (Set -database DATABASE in job file.)
-seg      Database segment group to use, e.g., with -dbqualitychannel. (Sets SEGMENT_GROUP in job file.)
-segfile  File with GPS segments (intervals) to use with SegToQC.tclsh to generate quality channel.
          If this option is given, output will be ascii2frame-P_SegToQC_segfile-gpsTime-deltaT.gwf.
          To load this file into a job use \%FILE(QC_FILE) in the job file framequery.
-scol     Column in segfile with GPS segment start time.
-ecol     Column in segfile with GPS segment end time.
-qch     Channel name to give to the quality channel generated by SetToQC.
-rmqc     Remove the file, ascii2frame-P_SegToQC_segfile-gpsTime-deltaT.gwf, after the job finishes
-st       GPS time to start the analysis. Set to "now" to start now (not yet implemented).
-dt       Duration in seconds to run each job on, and next job has start time st = st + dt.
-et       GPS time to end the analysis. Ends when time >= this time.
          Set to "endless" to run continuously. Do not set to run job once.
-e        Email address for messages from this script.
-log      Print output to specified log file.
-alog     Append to an existing log file.  (Otherwise if logfile exists it is overwritten.)
-blog     Create a backup copy the logfile upon exit. Backup name is logfile.timestamp.
-rec      Print 1 line record for each job run to specified record file.
-arec     Append to an existing record file.  (Otherwise if logfile exists it is overwritten.)
-brec     Create a backup copy the record file upon exit. Backup name is recordfile.timestamp.
-run      Run a command as LDAS job.
-job      File that sets up the command to run.

To see the command without running it, leave out the -run option.
If the run option is not given, the command is written to the log file,
but the job is not started or recorded.  This is noted in the log file.

To stop the script run the command: "touch stopjobdriver" from the script's pwd.
The script will stop after the next job finishes.

To restart the script from where it left off after the last record in a record file, use this syntax:

Syntax: ./DriveKPDJobs.tclsh [-v] [-run] -restart <recordfile> [-e <emailaddress>]
 -log <logfile> [-alog] [-blog] [-rec <recordfile>] [-arec] [-brec]

The site, frametype, interferometer, channel, database, seggroup, jobfile, deltaT, and endTime
fields will be read from the last record in the file given following the -restart option.
The startTime will be set to start time of last record + deltaT. Any of these field are
overwritten if you give the command line option for that field after the -restart option.
```

```
To redo jobs in a record file that finished with an error, use this syntax:

Syntax: ./DriveKPDJobs.tclsh [-v] [-run] -redo <recordfile> [-e <emailaddress>]
 -log <logfile> [-alog] [-blog] [-rec <recordfile>] [-arec] [-brec]

All jobs in the record file given after the redo option that finished with an error will be rerun.
```

Example of how to run the script to generate Sums for Stacks that are the PSDs, and Blocks are SFTs:

```
...
```

Example of how to run the script to generate Sums for Stacks that are the F-statistic, and Blocks are SFTs:

```
...
```

20.7 Schema Files

Below is a basic schema file used for testing the stackslide DSO in stand-alone mode. The -filterparams section can be modfied as described in the Filter Parameters sections above as long as the necessary data is in the file given after the -inputFile option.

Note that this needs to be updated.

```
 #
# lam boot schema for knownpulsardemod shared object
#
-np 2 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)"
-nodelist="(1-1)"
-dynlib="/home/gmendell/searchcode/lib/lalwrapper/libldasstackslide.so"
-dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)"
-filterparams="(733975200,0,4096,7200,118,60,60,1277.3,3.65,219,1,60,
1277.3,3.65,219,120,60,1277.3,3.65,219,H1,LHO,hwinject040903sig1,0,H1ASQ,SFT,
0,0,0,32,0,0,0,-1,0,0,0,0,0,1,1277.3,3.65,219,0,1,1,0,0,0,0,5.1471621319,
5.1471621319,0,1,0.3766960246,0.3766960246,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0)"
-realTimeRatio=0.9
-doLoadBalance=FALSE
-dataDistributor=W
-jobID=8
-inputFile="/PATHTODATAFILE/LDAS-WA949610_wrapperdata.ilwd"
```

20.8 References

[1] B. R. Brady and T. Creighton, gr-qc/9812014.

tracksearch

Index