Development of Dynamically Shared Objects for the LDAS wrapperAPI

Philip Charlton

LIGO Laboratory
California Institute of Technology

8 June 2001

LIGO-T010068-00-E

Contents

1 LDAS Dynamically-loaded Shared Objects

- A “dynamically-loaded shared object” (DSO) is a C library which can be opened by a running program.
- Developers write their DSO as a normal C library consisting of one or more source files and header files.
- Details of special compilation rules for making shared objects are handled by the `lalwrapper` installation.
- Main differences from a normal C library are:
  - Wrapper API will run *multiple instances* of the DSO in parallel.
  - Some *MPI communication* is necessary between instances.
  - Certain functions are *required* to be present.

  Developers must provide the required functions and MPI communication.
- A DSO can be tested by running it in “standalone mode” – does not require full build of LDAS or multi-node machine.
- Complete end-to-end test requires full installation of LDAS – output must be formatted to match database tables exactly.

2 Basic MPI functions

- Most MPI functions require an `MPI_comm` structure. In a wrapper DSO, one is supplied by the input arguments to `LALApplySearch`.
- Example: the function

  ```
  int MPI_Comm_size(MPI_Comm comm, int *psize)
  ```

  determines the size of the group associated with a communicator ie. the number of nodes in the group (including the search master but not wrapper master). This value is returned in the location pointed to by `psize`.
- LAL provides wrapper functions for many standard MPI functions, including message passing and data transmission.
- Not all MPI functions have LAL equivalents – some of these can be useful, others are restricted. Expert users may wish to apply them where appropriate.

2.1 LAL MPI message passing

- Calling

  ```
  void
  LALMPIRecvMsg (LALStatus  *status,
                 MPIMessage *msg,
                 MPI_Comm    mpiComm);
  ```

  causes a process to wait for messages – messages are received from *any* node that sends to the receiver.

- The `MPIMessage` structure is:

  ```
  typedef struct
  tagMPIMessage
  {
    INT4 msg;
    INT4 send;
    INT4 source;
  }
  MPIMessage;
  ```

- The `MPIMessage` is initially undefined. When `LALMPIRecvMsg` returns, `msg->source` contains the rank of the node that sent the message, and `msg->msg` contains a number which identifies the message type.

- Message type is just a number which can be interpreted or defined by the developer eg. via an `enum`.

- Message type tells the node what it should expect to receive next.

- Node rank tells it what node to listen to for the rest of the message.

- After initial `LALMPIRecvMsg`, receiver must specify the source it will receive the rest of the message from.

- Calling

  ```
  void
  LALMPISendMsg (
      LALStatus  *status,
      MPIMessage *msg,
      INT4        dest,
      MPI_Comm    mpiComm);
  ```

  causes a process to send the message contained in `msg` to the node number `dest`.

- Once communication is established, more data can be sent if required.
- LAL provides functions for sending and receiving most vector types eg.

  ```
  void
  LALMPISendREAL4Vector (LALStatus   *status,
                         REAL4Vector *vector,
                         INT4         dest,
                         MPI_Comm    mpiComm);

  void
  LALMPIRecvREAL4Vector (LALStatus   *status,
                         REAL4Vector *vector,
                         INT4         source,
                         MPI_Comm    mpiComm);
  ```

  See `lal/include/lal/Comm.h` for a complete list.

Simple example (recall – same code runs on all nodes):

```
MPIMessage mpiMsg;

if (master)
{
  /* Wait for messages from all nodes */
  LALMPIRecvMsg(status->statusPtr, &mpiMsg, comm);

  /* We have received something - see what it is */
  switch(mpiMsg.msg)
  {
  case 1:
    LALMPIRecvREAL4Vector(status->statusPtr, vec, mpiMsg.source, comm);
    break;
  case 2:
    /* ... etc */
    break;
  case 3:
    /* ... etc */
    break;
  default:
    /* possible error */
    break;
  }
}
else /* slave */
{
  mpiMsg.msg = 1;         /* Message ID */
  mpiMsg.source = myRank; /* My node rank */
  mpiMsg.send = 1;        /* Sending */

  /* Send message to node 0 (master) */
  LALMPISendMsg(status->statusPtr, &mpiMsg, 0, comm);
  LALMPISendREAL4Vector(status->statusPtr, vec, 0, comm);

  /* Finished that message now - carry on ... */
}
```

3 LDAS Flow of Control

- The four required functions are:

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

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

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

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

- Reason: Wrapper API is required to run arbitrary search codes but must be able to locate “entry points” into each library by name – every DSO must present the same interface to the wrapper. [1]

[1]For technical details, see `man` pages for dlopen() and dlysm().

- When a job is started, the mpiAPI starts instances of the wrapperAPI on the Beowulf.
- Each wrapper API loads the required DSO.
- Mpi API sends data to the "wrapper master" node on a socket.
- Wrapper master node broadcasts data to "search master" (and *optionally* to all slave nodes) via MPI communication.
- Depending on options, search master and slaves get identical data, or search master can be made responsible for sending data to slaves.
- Each instance of wrapper API calls
  - `LALInitSearch()`
  - `LALConditionData()`

  once.
- Each instance of wrapper API loops on
  - `LALApplySearch()`

  until the instance reports that it is finished. Results can be sent back in the output parameter when each call returns. Search master must report progress (fraction of work remaining) at the end of each call.
- Finally, each instance of wrapper API calls
  - `LALFinalizeSearch()`

  once.
- See figure `howtoFig-flowchart.pdf`.

3.1 Data required in standalone mode

- In standalone mode, data is read from a file which is formatted as Internal Light-weight Data (ILWD) eg. `example/wrapper.ilwd`:

```
<?ilwd?>
<ilwd name='ilwdConverter:container' size='3'>
  <ilwd name='ifodmro::sequence:primary' size='7'>
    <lstring name='real:domain' size='4'>TIME</lstring>
    <int_4u name='gps_sec:start_time' units='sec'>62348734</int_4u>
    <int_4u name='gps_nan:start_time' units='nanosec'>0</int_4u>
    <int_4u name='gps_sec:stop_time' units='sec'>62348738</int_4u>
    <int_4u name='gps_nan:stop_time' units='nanosec'>508789062</int_4u>
    <real_8 name='time:step_size' units='sec'>9.76562500e-04</real_8>
    <int_2s dims='4617' name='data' units='volts'>-250 366 -1024 ...
        ... -580 -654</int_2s>
    </ilwd>
    ...
</ilwd>
```

- Format is XML-like.
- ILWD objects can be nested.
- Tags specify attributes such as data type, name, size.
- After being read in, data is represented by a `multiDimData` structure (to be discussed below).
- File location and other details are specified in a `schema` file.

3.2 Example schema file: `moments.schema`

```
#
# lam boot schema for moments shared object
#
h -np 8 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-7)"
-dynlib="/home/charlton/local/lib/lalwrapper/libldasmoments.so"
-dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(0,0)"
-realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
-uniqueID=9.0 -inputFile="/home/charlton/local/share/lalwrapper/wrapper.ilwd"
```

Fields:

- `-np` – total number of nodes requested, including wrapper master and search master, leaving `(np - 2)` slaves.
- `-nodelist` – node numbers assigned to slaves.
- `-dynlib` – full path to DSO.
- `-filterparams=<string>` – DSO-specific parameters.
- `-realTimeRatio` – fraction of data duration which may be spent in this search eg. if 10 minutes of data is requested and this value is 0.9, the search must complete in 9 minutes.
- `-inputFile` – full path to data file.
- `-dataDistributor` – what nodes to broadcast initial data to. If `W`, identical data is sent to search master and all slaves. If `S`, data is sent only to search master – search master is responsible for sending data to other nodes.
- Remaining options are ignored in standalone mode.

4 Details of Required Functions

- Function prototypes and `typedefs` for structures are provided in `lalwrapper/include/LALWrapperInterface.h`.
- NOTE: Actual structure definitions are provided in `lalwrapper/include/wrapperInterfaceDatatypes.h`.
- Demonstrate usage by example: we provide a package `moments` which calculates different statistical moments (mean, variance, skewness, kurtosis) on different nodes. Package can be downloaded from `http://gravity.phys.psu.edu/LDASCamp/Friday/moments.tar`.
- Copy the `tar` file to `lalwrapper/contrib`. Untar it using `tar xvf moments.tar` – this adds a new source directory called `moments`.
- `Moments` package can be installed in a similar way to `helloworld` example. Need to add appropriate entries in `configure.in` and `contrib/Makefile.am` – see instructions in `moments/README.install`.
- After building, copy `moments/moments.schema` to `lalwrapper/example`, start LAM and run the DSO:

```
lamboot -v
mpirun moments.schema
```

- Output will be returned in file `output_1.ilwd`.
- This is just one example – others exist with different styles eg. `power`, `inspiral`, `fct`. Developers are free to write the DSO as they like provided the basic requirements are met.

4.1 `LALInitSearch`

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

Parameters:

- `status` – a LALStatus structure.
- `searchParams` – Pointer to a structure which must be defined by the developer. Value is initially undefined – should be allocated and initialized. This structure is provided to the other 3 functions.
- `initSearchParams` – alias for `InitParams` structure:

  ```
  typedef struct
  {
     INT4    argc;          /* arg count */
     CHAR**  argv;          /* args - argv[0] is "-filterparams" */
                            /* argv[1] is string after -filterparams */
     INT8    startTime;     /* seconds since January 1, 1970 */
     INT8    dataDuration;  /* duration of data up to nearest second */
     REAL4   realtimeRatio; /* from wrapperAPI command line */
     INT4    rank;          /* 0 if slave, 1 if searchMaster */
  } InitParams;
  ```

- Values passed in via `argc/argv` will need to be parsed.
- Note – no MPI communicator available.

Example: my search structure, defined in `Moments.h`:

```
typedef struct
tagHWSearchParams
{
  BOOLEAN searchMaster;
  INT4    rank;
  UINT4   numSlaves;
  UINT4   curSlaves;
  CHAR    channel[dbNameLimit];

  /* Container for the data received from the datacon API */
  REAL4Vector* vec;

  /* Container for sending/receiving results between nodes */
  REAL4Vector* res;

  /* The GPS start and end time of the data */
  gpsTimeInterval times;

  /*
   * The calculated statistics need to go in a structure that
   * persists between calls to ApplySearch.
   */

  REAL4 mean;
  REAL4 var;
  REAL4 skew;
  REAL4 kurt;

  /* Set to 1 when the structure is fully initialised */
  BOOLEAN initialised;

  /* This is only for debugging purposes */
  char id[IDLEN];
}
HWSearchParams;
```

My code, defined in `Moments.c`:

```
void LALInitSearch(
    LALStatus             *status,
    void                 **searchParams,
    LALInitSearchParams   *initSearchParams)
{
  HWSearchParams* const params = LALCalloc(1, sizeof(*params));

  INITSTATUS( status, "LALInitSearch", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  /* initialise parameters */
  params->searchMaster = initSearchParams->rank;
  params->rank         = 0;
  params->numSlaves    = 0;
  params->curSlaves    = 0;
  params->vec          = 0;
  params->res          = 0;
  params->initialised  = 0;

  strcpy(params->channel, "");

  params->mean = 0.0;
  params->var  = 0.0;
  params->skew = 0.0;
  params->kurt = 0.0;

  /* Set the output parameter to the new structure */
  *searchParams = params;

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

4.2 `LALConditionData`

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

Parameters:

- `inout` – alias for `inPut` structure:

  ```
  typedef struct
  {
     UINT4 numberSequences;   /* Number of sequences in input */
     stateVector* states;
     multiDimData* sequences; /* Array of sequences */
  } inPut;
  ```

  This is the structure containing the data read from file (in standalone mode) or datacon API (in LDAS).

- `searchParams` – pointer to the user-defined structure, which was allocated in `LALInitSearch()`.

- Note – no MPI communicator available in the present implementation. This will be changed in the near future to allow master to send data to slaves after conditioning.

Each data vector read from the ILWD file is presented as a `multiDimData` structure:

```
typedef struct
{
   CHAR name[maxMultiDimName];   /* Channel name from ILWD file */
   CHAR units[maxMultiDimUnits];
   domain space;                 /* Frequency or time domain */
   datatype type;                /* Enum type eg. int_2s, real_4 */
   interval range;               /* Time or frequency interval */
   UINT4 numberDimensions;
   UINT4* dimensions;            /* Array of dimensions */
   dcHistory* history;
   dataPointer data;             /* Union of different pointers */
                                 /* eg. INT2* int2s, REAL4* real4 */
} multiDimData;
```

- `LALConditionData` receives an array of `multiDimData`.
- There is a correspondence between attributes in the ILWD file and fields in the structure.
- See
  `api/wrapperAPI/so/src/wrapperInterfaceDatatypes.h`
  for full details of the structures used.

Example:

```
void
LALConditionData(
    LALStatus*      status,
    LALSearchInput* inout,
    void*           searchParams)
{
  HWSearchParams* const params = (HWSearchParams *) searchParams;
  BOOLEAN dataFound = 0;

  UINT4 i = 0;

  INITSTATUS( status, "LALConditionData", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  /* Create length 1 array to transmit/receive the result */
  LALSCreateVector(status->statusPtr, &(params->res), 1);

  for (i = 0; i < inout->numberSequences; ++i)
  {
    if (strstr(inout->sequences[i].name, "ifodmro"))
    {
      const INT2* dummyI2 = inout->sequences[i].data.int2s;
      const UINT4 nData = inout->sequences[i].dimensions[0];

      UINT4 k = 0;

      /*
       * All we want to do is convert the IFODMRO data from int to
       * float and get it to the ApplySearch routine
       */

      LALSCreateVector(status->statusPtr, &(params->vec), nData);

      /*
       * Copy from int2s to real4
       */
      for (k = 0; k < params->vec->length; ++k)
      {
        params->vec->data[k] = *dummyI2++;
      }

      strcpy(params->channel, "ifodmro");
      params->times = inout->sequences[i].range.dTime;

      dataFound = 1;
```

```
        break;
      }
    }

    /* oops - the required data was not found */
    if (!dataFound)
    {
      ABORT( status, CONDITIONDATA_EINPUT, CONDITIONDATA_MSGEINPUT);
    }

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

4.3 `LALApplySearch`

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

Parameters:

- `output` – alias for `SearchOutput` structure:

  ```
  typedef struct
  {
     INT4    numOutput;     /* number in the result array */
     outPut* result;        /* Array of outPut structures */
     REAL4   fracRemaining; /* fraction of search remaining */
     BOOLEAN notFinished;   /* 0 indicates that applySearch is finished */
  } SearchOutput;
  ```

- `input` – alias for `inPut` structure.
- `applyParams` – provides a pointer to the user-defined structure, which was allocated in `LALInitSearch()`, and the MPI parameters, including an MPI communicator.

  ```
  typedef struct
  tagLALApplySearchParams
  {
    LALMPIParams *mpiParams;
    void         *searchParams;
  }
  LALApplySearchParams;
  ```

Example:

```
void
LALApplySearch(
    LALStatus              *status,
    LALSearchOutput        *output,
    LALSearchInput         *input,
    LALApplySearchParams   *applyParams)
{
  HWSearchParams* const params = (HWSearchParams *) applyParams->searchParams;

  INITSTATUS( status, "LALApplySearch", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  if ((output == 0) || (input == 0) || (applyParams ==0))
  {
    ABORT(status, APPLYSEARCHH_ENULL, APPLYSEARCHH_MSGENULL);
  }

  /* Get the rank of this node - 0 for master, N > 0 for slave N */
  MPI_Comm_rank(*(applyParams->mpiParams->comm), &(params->rank));

  if (params->searchMaster) /* master */
  {
    ApplySearchMaster(status->statusPtr, output, input, params,
                      *(applyParams->mpiParams->comm));
  }
  else /* slave */
  {
    ApplySearchSlave(status->statusPtr, output, input, params,
                     *(applyParams->mpiParams->comm));
  }

  CHECKSTATUSPTR (status);

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

`ApplySearchMaster`:

- Calls `LALMPIRecvMessage` to wait for messages from slaves.
- On receiving, it determines what type of message this is – either transmission of a result or notification that a slave is finished.
- If a result, it calls `LALMPIRecvREAL4Vector` to retrieve the "vector" of results (actually just one value).
- If slave finished, nothing more needs to be received.
- Once all slaves are finished, the output is created (more on this later).
- An important point not shown in `moments` is that, generally, `ApplySearch()` on the master will have to wait for messages from each slave. This could be done in a loop, for example:

```
for (i = 0; i < numSlaves; ++i)
{
    /* Receive a message from each node,
       not necessarily in order */
    LALMPIRecvMsg(status->statusPtr, &mpiMsg, comm);

    /* Do rest of message processing ... */
}
```

`ApplySearchSlave`:

- Calculates a moment based on it's rank ie. node 1 calculates mean, node 2 calculates variance etc.
- Calls `LALMPISendMessage` to notify the master that it has a result.
- Calls `LALMPISendREAL4Vector` to send the result.
- Calls `LALMPISendMessage` to notify the master that it has finished.

4.4 Creating the output

- Output parameter from `LALApplySearch()` is

```
typedef struct
{
   INT4    numOutput;     /* number in the result array */
   outPut* result;        /* Array of outPut structures */
   REAL4   fracRemaining; /* fraction of search remaining */
   BOOLEAN notFinished;   /* 0 indicates that applySearch is finished */
} SearchOutput;
```

- MUST set `output->fracRemaining` to the "fraction of work remaining" at the end of `LALApplySearch`.
- `LALApplySearch` on master should arrange that $\sim$ 10% of the work is done for each call – `LALApplySearch` should be called $\sim$ 10 times.
- `output->fracRemaining` will be set to 0.0 when all is done.
- At the end of each `LALApplySearch()`, `numOutput` should be set to the number of elements in the `result` array (0 if no output).
- `outPut` structure has a number of fields to be filled:
- `result` is really an array of `outPut` structures:

```
typedef struct
{
   INT8 templateNumber;
   catagory search;
   BOOLEAN significant;
   stateVector* states;
   dataBase* results;
   multiDimData* optional;
} outPut;
```

  `numOutput` tells the wrapper API how many elements are in the array.

- `results` field is the head of a doubly-linked list of `dataBase` structures. This list must be populated with data for the database.
- Because of this the code for building output will have many lines but is just repetition – better to put this in a separate function `BuildOutput()`.

Example:

```
void
BuildOutput(LALStatus*            const status,
            const HWSearchParams* const params,
            LALSearchOutput*      const output)
{
  const CHAR* const PROGRAM = "Moments";

  dataBase* dbEntry = 0;
  outPut* outputArray = 0;

  INITSTATUS( status, "BuildOutput", MOMENTSC );

  /* Number of output objects */
  output->numOutput = 1;

  output->result = LALCalloc(1, sizeof(outPut));
  outputArray = output->result;

  outputArray[0].templateNumber = 0;     /* not relevant */
  outputArray[0].search         = burst;
  outputArray[0].significant    = 1;     /* 1 - IS significant */
  outputArray[0].states         = 0;
  outputArray[0].optional       = 0;

  /*
   * Start a linked list of database entries - the head of the list
   * starts with the "results" field
   */
  outputArray[0].results = LALCalloc(1, sizeof(dataBase));
  dbEntry = outputArray[0].results;
  dbEntry->previous = 0;

  /* Program name */
  snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
  snprintf(dbEntry->columnName, dbNameLimit, "program");

  dbEntry->type       = char_s_ptr;
  dbEntry->numberRows = 1;
  dbEntry->rows.chars = LALMalloc(sizeof(CHAR)*(strlen(PROGRAM) + 1) );
  strcpy(dbEntry->rows.chars, PROGRAM);
  dbEntry->rowDimensions = LALMalloc(sizeof(UINT4));
  /* Don’t include terminating null */
  dbEntry->rowDimensions[0] = strlen(PROGRAM);

  dbEntry->next = LALCalloc(1, sizeof(dataBase));
  dbEntry->next->previous = dbEntry;
  dbEntry = dbEntry->next;

  /* Mean */
  snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
  snprintf(dbEntry->columnName, dbNameLimit, "mean");

  dbEntry->type       = real_8;
  dbEntry->numberRows = 1;
  dbEntry->rows.real8 = LALMalloc( sizeof(REAL8) );
  *(dbEntry->rows.real8) = params->mean;
  dbEntry->rowDimensions = NULL;

  /* ... etc */

  /* Terminate the list */
  dbEntry->next = 0;

  RETURN(status);
}
```

4.5 LALFinalizeSearch

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

Example:

```
void
LALFinalizeSearch(
    LALStatus             *status,
    void                 **searchParams)
{
  HWSearchParams* const params = (HWSearchParams *) *searchParams;

  INITSTATUS( status, "LALFinalizeSearch", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  LALSDestroyVector(status->statusPtr, &(params->vec));
  LALSDestroyVector(status->statusPtr, &(params->res));

  LALFree(*searchParams);

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

5 Suggested exercises

- Install and build the `helloworld` package.
- Add a structure to `HelloWorld.h` to hold search-specific information, such as the node rank. Allocate it in `LALApplySearch()` and free it in `LALFinalizeSearch()`.
- Use `MPI_Comm_size()` to determine each node's rank. Print the rank.
- Have the master receive a message in `LALApplySearch()`. Have the slaves send a message to the master.
- Create a `REAL4Vector` in `LALInitSearch()` (remember to destroy it in `LALFinalizeSearch()`).
- Have the slaves put some values in the vector and send it to the master. Have the master receive the vector.
- Add a function to build up an output structure. Create a linked-list of database elements consisting of one (or more) elements and see that they printed out in the `output_1.ilwd` file (to start with, the fields could have fixed values).
- Make a result sent from a slave part of the output.
- Install and build the `moments` package.
- In the base code, the master only receives a single message from one slave at each iteration of `ApplySearch()`. Add a loop to `ApplySearchMaster()` so that it receives a message from all slaves before proceeding.
- Incorporate a LAL function of your own devising into `moments`.

A lal/include/Comm.h

```
/*-----------------------------------------------------------------------
 *
 * File Name: Comm.h
 *
 * Author: Allen, B., Brown D. A. and Creighton, J. D. E.
 *
 * Revision: $Id: Comm.h,v 1.7 2001/04/12 03:17:00 duncan Exp $
 *
 *-----------------------------------------------------------------------
 */

#ifndef _COMM_H
#define _COMM_H

#include <mpi.h>
#include <lal/LALDatatypes.h>
#include <lal/AVFactories.h>

#ifdef  __cplusplus
extern "C" {
#endif

NRCSID (COMMH, "$Id: Comm.h,v 1.7 2001/04/12 03:17:00 duncan Exp $");

#define COMM_ENULL 1
#define COMM_ENNUL 2
#define COMM_ESIZE 3
#define COMM_ESTRL 4
#define COMM_EMPIE 5
#define COMM_ESENV 6
#define COMM_ESYSC 7
#define COMM_ENOBJ 8
#define COMM_EHAND 9

#define COMM_MSGENULL "Null pointer"
#define COMM_MSGENNUL "Non-Null pointer"
#define COMM_MSGESIZE "Invalid size"
#define COMM_MSGESTRL "String too long"
#define COMM_MSGEMPIE "MPI error"
#define COMM_MSGESENV "Couldn’t set environment variable"
#define COMM_MSGESYSC "Error executing system command"
#define COMM_MSGENOBJ "Invalid number of objects"
#define COMM_MSGEHAND "Wrong handshake"

/* Structure for identifying processors */
typedef struct
tagMPIId
{
  INT4 numProcs;
  INT4 myId;
  INT4 nameLen;
  CHAR procName[MPI_MAX_PROCESSOR_NAME];
}
MPIId;

typedef struct
tagMPIDebugParams
{
  const CHAR *debugger;
  CHAR *progName;
  INT4  delay;
  INT4  myId;
  MPI_Comm mpiComm;
}
MPIDebugParams;

typedef enum
{
  MPIDone,
  MPIMisc,
  MPIErr,
  MPIMsg,
  MPICHARVector,
  MPICHARVectorData,
  MPII2Vector,
  MPII2VectorData,
  MPII4Vector,
  MPII4VectorData,
  MPII8Vector,
  MPII8VectorData,
  MPISVector,
  MPISVectorData,
  MPIDVector,
  MPIDVectorData,
  MPICVector,
  MPICVectorData,
  MPIZVector,
  MPIZVectorData,
  MPII2TimeSeries,
  MPII4TimeSeries,
  MPII8TimeSeries,
  MPISTimeSeries,
  MPIDTimeSeries,
  MPICTimeSeries,
  MPIZTimeSeries,
  MPII2FrequencySeries,
  MPII4FrequencySeries,
  MPII8FrequencySeries,
  MPISFrequencySeries,
  MPIDFrequencySeries,
  MPICFrequencySeries,
  MPIZFrequencySeries
}
MPIMsgCode;

typedef struct
```

```
tagMPIMessage
{
  INT4 msg;
  INT4 send;
  INT4 source;
}
MPIMessage;

typedef struct
tagExchParams
{
  INT4           send;
  INT4           numObjects;
  INT4           partnerProcNum;
  INT4           myProcNum;
  INT4           exchObjectType;
  MPI_Comm       mpiComm;
}
ExchParams;

typedef struct
tagInitExchParams
{
  INT4           myProcNum;
  MPI_Comm       mpiComm;
}
InitExchParams;



void
LALMPIExportEnvironment (
    LALStatus     *status,
    const CHAR *env,
    INT4        myId,
    MPI_Comm    mpiComm
    );

void
LALMPIDebug (
    LALStatus         *status,
    MPIDebugParams *params
    );

void
LALMPIKillScript (
    LALStatus  *status,
    MPIId   *id,
    MPI_Comm    mpiComm
    );


void
LALMPISendMsg (
    LALStatus     *status,
    MPIMessage *msg,
    INT4        dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvMsg (
    LALStatus     *status,
    MPIMessage *msg,
    MPI_Comm    mpiComm
    );

void
LALMPISendCHARVector (
    LALStatus     *status,
    CHARVector *vector,
    INT4        dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvCHARVector (
    LALStatus     *status,
    CHARVector *vector,
    INT4        source,
    MPI_Comm    mpiComm
    );

void
LALMPISendINT2Vector (
    LALStatus     *status,
    INT2Vector *vector,
    INT4        dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvINT2Vector (
    LALStatus     *status,
    INT2Vector *vector,
    INT4        source,
    MPI_Comm    mpiComm
    );

void
LALMPISendINT4Vector (
    LALStatus     *status,
    INT4Vector *vector,
    INT4        dest,
    MPI_Comm    mpiComm
    );

void
```

```
LALMPIRecvINT4Vector (
    LALStatus     *status,
    INT4Vector *vector,
    INT4        source,
    MPI_Comm    mpiComm
    );

void
LALMPISendREAL4Vector (
    LALStatus      *status,
    REAL4Vector *vector,
    INT4         dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvREAL4Vector (
    LALStatus      *status,
    REAL4Vector *vector,
    INT4         source,
    MPI_Comm    mpiComm
    );

void
LALMPISendCOMPLEX8Vector (
    LALStatus         *status,
    COMPLEX8Vector *vector,
    INT4            dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvCOMPLEX8Vector (
    LALStatus         *status,
    COMPLEX8Vector *vector,
    INT4            source,
    MPI_Comm    mpiComm
    );

void
LALMPISendINT2TimeSeries (
    LALStatus         *status,
    INT2TimeSeries *series,
    INT4            dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvINT2TimeSeries (
    LALStatus         *status,
    INT2TimeSeries *series,
    INT4            source,
    MPI_Comm    mpiComm
    );
```

```
void
LALMPISendREAL4TimeSeries (
    LALStatus          *status,
    REAL4TimeSeries *series,
    INT4             dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvREAL4TimeSeries (
    LALStatus          *status,
    REAL4TimeSeries *series,
    INT4             source,
    MPI_Comm    mpiComm
    );

void
LALMPISendCOMPLEX8TimeSeries (
    LALStatus             *status,
    COMPLEX8TimeSeries *series,
    INT4                dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvCOMPLEX8TimeSeries (
    LALStatus             *status,
    COMPLEX8TimeSeries *series,
    INT4                source,
    MPI_Comm    mpiComm
    );

void
LALMPISendREAL4FrequencySeries (
    LALStatus               *status,
    REAL4FrequencySeries *series,
    INT4                  dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvREAL4FrequencySeries (
    LALStatus               *status,
    REAL4FrequencySeries *series,
    INT4                  source,
    MPI_Comm    mpiComm
    );

void
LALMPISendCOMPLEX8FrequencySeries (
    LALStatus                  *status,
    COMPLEX8FrequencySeries *series,
    INT4                     dest,
    MPI_Comm    mpiComm
    );

void
LALMPIRecvCOMPLEX8FrequencySeries (
    LALStatus                  *status,
    COMPLEX8FrequencySeries *series,
    INT4                     source,
    MPI_Comm    mpiComm
    );

void
LALInitializeExchange (
    LALStatus      *status,
    ExchParams **exchParamsOut,
    ExchParams  *exchParamsInp,
    InitExchParams *params
    );

void
LALFinalizeExchange (
    LALStatus      *status,
    ExchParams **exchParams
    );

void
LALExchangeUINT4 (
    LALStatus         *status,
    UINT4             *object,
    ExchParams        *exchParms
                 );

#ifdef  __cplusplus
}
#endif

#endif /* _COMM_H */
```

B lalwrapper/include/wrapperInterfaceDatatypes.h

```
/* $Id: wrapperInterfaceDatatypes.h,v 1.6 2001/05/15 22:42:21 duncan Exp $ */

#ifndef WRAPPER_INTERFACE_DATATYPES_H
#define WRAPPER_INTERFACE_DATATYPES_H

/* MPI Header File */
#include <mpi.h>

/* Local Header Files */
#include <lal/LALAtomicDatatypes.h>


#ifdef __cplusplus
extern "C"{
#endif


/* Domain type */
typedef enum
{
   timeD,
   freqD,
   bothD
} domain;


/* Datatype */
typedef enum
{
   boolean_lu,
   char_s,
   char_u,
   int_2s,
   int_2u,
   int_4s,
   int_4u,
   int_8s,
   int_8u,
   real_4,
   real_8,
   complex_8,
   complex_16,
   char_s_ptr,
   char_u_ptr
}datatype;


/* Pointer type checking */
typedef union
{
   BOOLEAN*   boolean;
   CHAR*      chars;
   UCHAR*     charu;
   INT2*      int2s;
   UINT2*     int2u;
   INT4*      int4s;
   UINT4*     int4u;
   INT8*      int8s;
   UINT8*     int8u;
   REAL4*     real4;
   REAL8*     real8;
   COMPLEX8*  complex8;
   COMPLEX16* complex16;
}dataPointer;


/* Time domain interval */
typedef struct
{
   UINT8 numberSamples;
   UINT4 startSec;
   UINT4 startNan;
   UINT4 stopSec;
   UINT4 stopNan;
   REAL8 timeStepSize;
}gpsTimeInterval;


/* Frequency domain interval */
typedef struct
{
   UINT8 numberSamples;
   UINT4 gpsStartTimeSec;
   UINT4 gpsStartTimeNan;
   UINT4 gpsStopTimeSec;
   UINT4 gpsStopTimeNan;
   REAL8 startFreq;
   REAL8 stopFreq;
   REAL8 freqStepSize;
}frequencyInterval;


/* Time & Frequency domain interval */
typedef struct
{
   UINT8 numberSamples;
   UINT4 gpsStartTimeSec;
   UINT4 gpsStartTimeNan;
   UINT4 gpsStopTimeSec;
   UINT4 gpsStopTimeNan;
   REAL8 startFreq;
   REAL8 stopFreq;
   REAL8 timeStepSize;
   REAL8 freqStepSize;
}timeFreqInterval;
```

```
/* Interval domain */
typedef union
{
   gpsTimeInterval dTime;
   frequencyInterval dFreq;
   timeFreqInterval dBoth;
}interval;


#define maxHistoryName  64
#define maxHistoryUnits 64


/* History linked list */
typedef struct dcHistoryTag
{
   struct dcHistoryTag* previous;
   CHAR name[maxHistoryName];
   CHAR units[maxHistoryUnits];
   datatype type;
   UINT4 numberValues;
   dataPointer value;
   struct dcHistoryTag* next;
}dcHistory;

#define maxMultiDimName  256
#define maxMultiDimUnits 256


/* Multidimensional data */
typedef struct
{
   CHAR name[maxMultiDimName];
   CHAR units[maxMultiDimUnits];
   domain space;
   datatype type;
   interval range;
   UINT4 numberDimensions;
   UINT4* dimensions;
   dcHistory* history;
   dataPointer data;
}multiDimData;


#define maxStateName 64


/* State vector */
typedef struct stateVectorTag
{
   struct stateVectorTag* previous;
   CHAR stateName[maxStateName];
   multiDimData* store;
   struct stateVectorTag* next;
}stateVector;


/* Input data structure */
typedef struct
{
   UINT4 numberSequences;
   stateVector* states;
   multiDimData* sequences;
}inPut;


/* Astrophysical/instrumental search catagories */
typedef enum
{
   binaryInspiral,
   ringDown,
   periodic,
   burst,
   stocastic,
   timeFreq,
   instrumental,
   protoType,
   experimental
}catagory;


#define dbNameLimit 19


/* Database structure */
typedef struct dataBaseTag
{
   struct dataBaseTag* previous;
   CHAR tableName[dbNameLimit];
   CHAR columnName[dbNameLimit];
   datatype type;
   UINT4 numberRows;
   dataPointer rows;
   UINT4* rowDimensions;   /* Non zero only for char_s_ptr or char_u_ptr */
   struct dataBaseTag* next;
}dataBase;


/* Output data structure */
typedef struct
{
   INT8 templateNumber;
   catagory search;
   BOOLEAN significant;
   stateVector* states;
   dataBase* results;
   multiDimData* optional;
```

```
}outPut;


/* Structures specific to the search functions */

typedef struct
{
   INT4    argc;
   CHAR**  argv;
   INT8    startTime;     /* seconds since January 1, 1970 */
   INT8    dataDuration;  /* length of data rounded up to nearest second */
   REAL4   realtimeRatio; /* from wrapperAPI command line */
   INT4    rank;          /* -1 if wrapperMaster, 0 if slave, 1 if searchMaster */
}InitParams;


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


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


typedef struct
{
   INT4    numOutput;
   outPut* result;
   REAL4   fracRemaining; /* fraction of search remaining */
   BOOLEAN notFinished;   /* false indicates that applySearch is finished */
}SearchOutput;


#ifdef __cplusplus
}
#endif


#endif
```

C Moments.h

```
/* -*- mode: c; c-basic-offset: 2; -*- */
/******** <lalVerbatim file="MomentsHV"> ********
Author: Charlton, P; Brady, P R
$Id: $
********* </lalVerbatim> ********/

#ifndef _MOMENTS_H
#define _MOMENTS_H

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

NRCSID( MOMENTSH, "$Id: ");

/******************************** <lalErrTable file="MomentsHErrTab"> */

#define CONDITIONDATA_ENULL     1
#define CONDITIONDATA_ENNUL     2
#define CONDITIONDATA_EALOC     3
#define CONDITIONDATA_EARGS     4
#define CONDITIONDATA_EINPUT    5
#define CONDITIONDATA_EDATZ     6

#define CONDITIONDATA_MSGENULL   "Null pointer"
#define CONDITIONDATA_MSGENNUL   "Non-null pointer"
#define CONDITIONDATA_MSGEALOC   "Memory allocation error"
#define CONDITIONDATA_MSGEARGS   "Wrong number of arguments"
#define CONDITIONDATA_MSGEINPUT  "Wrong data names in input structure"
#define CONDITIONDATA_MSGEDATZ   "Got less data than expected"

#define APPLYSEARCHH_ENULL 1
#define APPLYSEARCHH_ENNUL 2
#define APPLYSEARCHH_EALOC 3
#define APPLYSEARCHH_ENUMZ 4
#define APPLYSEARCHH_EDELT 8
#define APPLYSEARCHH_ERANK 9
#define APPLYSEARCHH_EUEXT 10

#define APPLYSEARCHH_MSGENULL "Null pointer"
#define APPLYSEARCHH_MSGENNUL "Non-null pointer"
#define APPLYSEARCHH_MSGEALOC "Memory allocation error"
#define APPLYSEARCHH_MSGENUMZ "Data segment length is zero"
#define APPLYSEARCHH_MSGEDELT "deltaT is zero or negative"
#define APPLYSEARCHH_MSGERANK "Search node has incorrect rank"
#define APPLYSEARCHH_MSGEUEXT "Unrecognised exchange type"

/************************************ </lalErrTable> */

/* Identifiers for the types of messages we will be sending/receiving */
enum
{
  ExchMean,
  ExchVar,
  ExchSkew,
  ExchKurt,
  ExchHOM,
  ExchFinished
}
ExchType;

#define IDLEN 20

/*
 * The user-defined structure which holds data relevant to our particular
 * wrapper shared object
 */
typedef struct
tagHWSearchParams
{
  BOOLEAN searchMaster;
  INT4    rank;
  UINT4   numSlaves;
  UINT4   curSlaves;
  CHAR    channel[dbNameLimit];

  /* Container for the data received from the datacon API */
  REAL4Vector* vec;

  /* Container for sending/receiving results between nodes */
  REAL4Vector* res;

  /* The GPS start and end time of the data */
  gpsTimeInterval times;

  /*
   * The calculated statistics need to go in a structure that
   * persists between calls to ApplySearch.
   */

  REAL4 mean;
  REAL4 var;
  REAL4 skew;
  REAL4 kurt;

  /* Set to 1 when the structure is fully initialised */
  BOOLEAN initialised;

  /* This is only for debugging purposes */
  char id[IDLEN];
}
HWSearchParams;

#endif /* _MOMENTS_H */
```

D Moments.c

```
/* -*- mode: c; c-basic-offset: 2; -*- */
/******** <lalVerbatim file="MomentsCV"> ********
Author: Charlton, Philip; Brady, P R
$Id: MomentsC.tex,v 1.1 2001/04/26 15:24:21 patrick Exp $
********* </lalVerbatim> ********/

//#define DEBUG_MOMENTS
#define USE_LDASCAMPMOMENT

#ifdef DEBUG_MOMENTS
#include <stdio.h>
#endif

#include <math.h>

#include <lal/LALStdlib.h>
#include <lal/AVFactories.h>

#ifdef USE_LDASCAMPMOMENT
#include <lal/LDASCampMoment.h>
#endif

/* LAL MPI interface */
#include <lal/Comm.h>

#include <Moments.h>

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

/********************* local static function declarations *******************/

static
void
BuildOutput(LALStatus*            const status,
            const HWSearchParams* const params,
            LALSearchOutput*      const output);

static
void
ApplySearchMaster(
    LALStatus*            const status,
    LALSearchOutput*      const output,
    const LALSearchInput* const input,
    HWSearchParams*       const params,
    MPI_Comm              const comm);

static
void
ApplySearchSlave(
    LALStatus*            const status,
    LALSearchOutput*      const output,
    const LALSearchInput* const input,
    HWSearchParams*       const params,
    MPI_Comm                    comm);

/************************* wrapperAPI function definitions  *****************/

/*
 * Function LALInitSearch
 *
 * Initialise the search structures
 */
void LALInitSearch(
    LALStatus             *status,
    void                 **searchParams,
    LALInitSearchParams   *initSearchParams)
{
  HWSearchParams* const params = LALCalloc(1, sizeof(*params));

  INITSTATUS( status, "LALInitSearch", MOMENTSC );
  ATTATCHSTATUSPTR (status);

#ifdef DEBUG_MOMENTS
  fprintf(stdout, "LALInitSearch\n");
  fflush(stdout);
#endif

  /* initialise parameters */
  params->searchMaster = initSearchParams->rank;
  params->rank         = 0;
  params->numSlaves    = 0;
  params->curSlaves    = 0;
  params->vec          = 0;
  params->res          = 0;
  params->initialised  = 0;

  strcpy(params->channel, "");

  params->mean = 0.0;
  params->var  = 0.0;
  params->skew = 0.0;
  params->kurt = 0.0;

  /* Set the output parameter to the new structure */
  *searchParams = params;

  DETATCHSTATUSPTR (status);
  RETURN(status);
}

/*
 * Function LALConditionData
 *
 * Reads the input data provided to the wrapperAPI from LDAS (usually
 * from the data-conditioning API) which is a list of sequences.
 * The code is looking for time-series data with the name "H2:LSC-AS_Q"
 * or "ifodmro". When found, it copies it to a LAL REAL4Vector for later
 * use and also notes the GPS start-time of the sequence.
 *
 */
void
LALConditionData(
    LALStatus*      status,
    LALSearchInput* inout,
    void*           searchParams)
{
  HWSearchParams* const params = (HWSearchParams *) searchParams;
  BOOLEAN dataFound = 0;

  UINT4 i = 0;

  INITSTATUS( status, "LALConditionData", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  /* Create length 1 array to transmit/receive the result */
  LALSCreateVector(status->statusPtr, &(params->res), 1);

  for (i = 0; i < inout->numberSequences; ++i)
  {
    if (strstr(inout->sequences[i].name, "H2\\:LSC-AS_Q"))
    {
      const UINT4 nData = inout->sequences[i].dimensions[0];

      LALSCreateVector(status->statusPtr, &(params->vec), nData);

      memcpy(params->vec->data, inout->sequences[i].data.real4,
             sizeof(REAL4)*nData);

      strcpy(params->channel, "H2:LSC-AS_Q");
      params->times = inout->sequences[i].range.dTime;

      dataFound = 1;
      break;
    }
    else if (strstr(inout->sequences[i].name, "ifodmro"))
    {
      const INT2* dummyI2 = inout->sequences[i].data.int2s;
      const UINT4 nData = inout->sequences[i].dimensions[0];

      UINT4 k = 0;

      /*
       * All we want to do is convert the IFODMRO data from int to
       * float and get it to the ApplySearch routine
       */

      LALSCreateVector(status->statusPtr, &(params->vec), nData);

      /*
       * Copy from int2s to real4
```

```
         */
        for (k = 0; k < params->vec->length; ++k)
        {
          params->vec->data[k] = *dummyI2++;
        }

        strcpy(params->channel, "ifodmro");
        params->times = inout->sequences[i].range.dTime;

        dataFound = 1;
        break;
      }
    }

    /* oops - the required data was not found */
    if (!dataFound)
    {
      ABORT( status, CONDITIONDATA_EINPUT, CONDITIONDATA_MSGEINPUT);
    }

    DETATCHSTATUSPTR (status);
    RETURN(status);
  }

  /*
   * Function LALApplySearch
   *
   * Determines if this node is a master or slave and calls the
   * appropriate function.
   */
  void
  LALApplySearch(
      LALStatus              *status,
      LALSearchOutput        *output,
      LALSearchInput         *input,
      LALApplySearchParams   *applyParams)
  {
    HWSearchParams* const params = (HWSearchParams *) applyParams->searchParams;

    INITSTATUS( status, "LALApplySearch", MOMENTSC );
    ATTATCHSTATUSPTR (status);

    if ((output == 0) || (input == 0) || (applyParams ==0))
    {
      ABORT(status, APPLYSEARCHH_ENULL, APPLYSEARCHH_MSGENULL);
    }

    MPI_Comm_rank(*(applyParams->mpiParams->comm), &(params->rank));

    if (params->searchMaster) /* master */
    {
      ApplySearchMaster(status->statusPtr, output, input, params,
                        *(applyParams->mpiParams->comm));
    }
    else /* slave */
    {
      ApplySearchSlave(status->statusPtr, output, input, params,
                       *(applyParams->mpiParams->comm));
    }

    CHECKSTATUSPTR (status);

    DETATCHSTATUSPTR (status);
    RETURN(status);
  }

  /*
   * Function LALFinalizeSearch
   *
   * Finalize routine does nothing but check arguments and free memory
   */
  void
  LALFinalizeSearch(
      LALStatus             *status,
      void                 **searchParams)
  {
    HWSearchParams* const params = (HWSearchParams *) *searchParams;

    INITSTATUS( status, "LALFinalizeSearch", MOMENTSC );
    ATTATCHSTATUSPTR (status);

  #ifdef DEBUG_MOMENTS
    fprintf(stdout, "%s: LALFinalizeSearch\n", params->id);
    fflush(stdout);
  #endif

    LALSDestroyVector(status->statusPtr, &(params->vec));
    LALSDestroyVector(status->statusPtr, &(params->res));

    LALFree(*searchParams);

    DETATCHSTATUSPTR (status);
    RETURN(status);
  }

  /********************* local static function definitions *******************/

  /*
   * Function BuildOutput
   *
   * Create a linked list containing the database entries and put it in
   * the search output structure
   */
  static
  void
  BuildOutput(LALStatus*            const status,
              const HWSearchParams* const params,
              LALSearchOutput*      const output)
```

```
{
  /*
   * The program name which appears in the PROGRAM field of
   * the SUMM_STATISTICS table
   */
  const CHAR* const PROGRAM = "Moments";

  dataBase* dbEntry = 0;
  outPut* outputArray = 0;

  INITSTATUS( status, "BuildOutput", MOMENTSC );

  /* Number of output objects */
  output->numOutput = 1;

  output->result = LALCalloc(1, sizeof(outPut));
  outputArray = output->result;

  outputArray[0].templateNumber = 0;     /* not relevant */
  outputArray[0].search         = burst;
  outputArray[0].significant    = 1;     /* 1 - IS significant */
  outputArray[0].states         = 0;
  outputArray[0].optional       = 0;

  /*
   * Start a linked list of database entries - the head of the list
   * starts with the "results" field
   */
  outputArray[0].results = LALCalloc(1, sizeof(dataBase));
  dbEntry = outputArray[0].results;
  dbEntry->previous = 0;

  /* Program name */
  snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
  snprintf(dbEntry->columnName, dbNameLimit, "program");

  dbEntry->type       = char_s_ptr;
  dbEntry->numberRows = 1;
  dbEntry->rows.chars = LALMalloc(sizeof(CHAR)*(strlen(PROGRAM) + 1) );
  strcpy(dbEntry->rows.chars, PROGRAM);
  dbEntry->rowDimensions = LALMalloc(sizeof(UINT4));
  /* Don’t include terminating null */
  dbEntry->rowDimensions[0] = strlen(PROGRAM);

  dbEntry->next = LALCalloc(1, sizeof(dataBase));
  dbEntry->next->previous = dbEntry;
  dbEntry = dbEntry->next;

  /* Start time */
  snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
  snprintf(dbEntry->columnName, dbNameLimit, "start_time");

  dbEntry->type       = int_4s;
  dbEntry->numberRows = 1;
```

```
dbEntry->rows.int4s = LALMalloc( sizeof(INT4) );
*(dbEntry->rows.int4s) = params->times.startSec;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "start_time_ns");

dbEntry->type       = int_4s;
dbEntry->numberRows = 1;
dbEntry->rows.int4s = LALMalloc( sizeof(INT4) );
*(dbEntry->rows.int4s) = params->times.startNan;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

/* End time */
snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "end_time");

dbEntry->type       = int_4s;
dbEntry->numberRows = 1;
dbEntry->rows.int4s = LALMalloc( sizeof(INT4) );
*(dbEntry->rows.int4s) = params->times.stopSec;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "end_time_ns");

dbEntry->type       = int_4s;
dbEntry->numberRows = 1;
dbEntry->rows.int4s = LALMalloc( sizeof(INT4) );
*(dbEntry->rows.int4s) = params->times.stopNan;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

/* Number of samples */
snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "samples");

dbEntry->type       = int_4s;
dbEntry->numberRows = 1;
```

```
dbEntry->rows.int4s = LALMalloc( sizeof(INT4) );
*(dbEntry->rows.int4s) = params->times.numberSamples;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

/* Channel name */
snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "channel");

dbEntry->type       = char_s_ptr;
dbEntry->numberRows = 1;
dbEntry->rows.chars = LALMalloc(sizeof(CHAR)*(strlen(params->channel) + 1) );
strcpy(dbEntry->rows.chars, params->channel);
dbEntry->rowDimensions = LALMalloc(sizeof(UINT4));
/* Don’t include terminating null */
dbEntry->rowDimensions[0] = strlen(params->channel);

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

/* Mean */
snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "mean");

dbEntry->type       = real_8;
dbEntry->numberRows = 1;
dbEntry->rows.real8 = LALMalloc( sizeof(REAL8) );
*(dbEntry->rows.real8) = params->mean;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

/* Variance */
snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
snprintf(dbEntry->columnName, dbNameLimit, "variance");

dbEntry->type       = real_8;
dbEntry->numberRows = 1;
dbEntry->rows.real8 = LALMalloc( sizeof(REAL8) );
*(dbEntry->rows.real8) = params->var;
dbEntry->rowDimensions = NULL;

dbEntry->next = LALCalloc(1, sizeof(dataBase));
dbEntry->next->previous = dbEntry;
dbEntry = dbEntry->next;

/* Skewness */
snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
```

```
  snprintf(dbEntry->columnName, dbNameLimit, "skewness");

  dbEntry->type       = real_8;
  dbEntry->numberRows = 1;
  dbEntry->rows.real8 = LALMalloc( sizeof(REAL8) );
  *(dbEntry->rows.real8) = params->skew;
  dbEntry->rowDimensions = NULL;

  dbEntry->next = LALCalloc(1, sizeof(dataBase));
  dbEntry->next->previous = dbEntry;
  dbEntry = dbEntry->next;

  /* Kurtosis */
  snprintf(dbEntry->tableName, dbNameLimit, "summ_statistics");
  snprintf(dbEntry->columnName, dbNameLimit, "kurtosis");

  dbEntry->type       = real_8;
  dbEntry->numberRows = 1;
  dbEntry->rows.real8 = LALMalloc( sizeof(REAL8) );
  *(dbEntry->rows.real8) = params->kurt;
  dbEntry->rowDimensions = NULL;

  /* Terminate the list */
  dbEntry->next = 0;

  RETURN(status);
}

/*
 * Function ApplySearchMaster
 *
 * Called from LALApplySearch only on the search master node. It’s
 * main job is to wait for responses for each slave ie. retrieve the
 * statistics and wait for the slaves to end, then make the output.
 */
static
void
ApplySearchMaster(
    LALStatus*            const status,
    LALSearchOutput*      const output,
    const LALSearchInput* const input,
    HWSearchParams*       const params,
    MPI_Comm              const comm)
{
  MPIMessage mpiMsg;

  INITSTATUS( status, "ApplySearchMaster", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  /* only do this the first time the master is called */
  if (params->initialised == 0)
  {
    /* a prefix to identify debugging messages */
    snprintf(params->id, IDLEN, "master[node %2d]", params->rank);
    /* get the number of search slaves */
    MPI_Comm_size(comm, &(params->numSlaves));
    /* The number returned INCLUDES the search master, so subtract 1 */
    params->numSlaves -= 1;
    params->curSlaves = params->numSlaves;

    params->initialised = 1;
  }

#ifdef DEBUG_MOMENTS
  fprintf(stdout, "%s: Current umber of slaves: %u\n",
          params->id, params->curSlaves);
  fflush(stdout);
#endif

  if (params->curSlaves != 0)
  {
    /* wait for a message */
    LALMPIRecvMsg(status->statusPtr, &mpiMsg, comm);
    CHECKSTATUSPTR (status);

    /* determine the message type set in the "msg" field */
    switch (mpiMsg.msg)
    {
      /*
       *  One of the moments is ready
       */
    case ExchMean:
      LALMPIRecvREAL4Vector(status->statusPtr, params->res,
                            mpiMsg.source, comm);
      CHECKSTATUSPTR (status);
      params->mean = params->res->data[0];
#ifdef DEBUG_MOMENTS
      fprintf(stdout, "%s: Received mean with value %f\n", params->id,
              params->mean);
      fflush(stdout);
#endif
      break;

    case ExchVar:
      LALMPIRecvREAL4Vector(status->statusPtr, params->res,
                            mpiMsg.source, comm);
      CHECKSTATUSPTR (status);
      params->var = params->res->data[0];
#ifdef DEBUG_MOMENTS
      fprintf(stdout, "%s: Received variance with value %f\n", params->id,
              params->var);
      fflush(stdout);
#endif
      break;

    case ExchSkew:
      LALMPIRecvREAL4Vector(status->statusPtr, params->res,
                          mpiMsg.source, comm);
      CHECKSTATUSPTR (status);
      params->skew = params->res->data[0];
#ifdef DEBUG_MOMENTS
      fprintf(stdout, "%s: Received skewness with value %f\n", params->id,
              params->skew);
      fflush(stdout);
#endif
      break;

    case ExchKurt:
      LALMPIRecvREAL4Vector(status->statusPtr, params->res,
                          mpiMsg.source, comm);
      CHECKSTATUSPTR (status);
      params->kurt = params->res->data[0];
#ifdef DEBUG_MOMENTS
      fprintf(stdout, "%s: Received kurtosis with value %f\n", params->id,
              params->kurt);
      fflush(stdout);
#endif
      break;

    case ExchHOM:
      /* Higher order moments are collected but ignored */
      LALMPIRecvREAL4Vector(status->statusPtr, params->res,
                          mpiMsg.source, comm);
      CHECKSTATUSPTR (status);
#ifdef DEBUG_MOMENTS
      fprintf(stdout, "%s: Received higher-order moment with value %f\n",
              params->id, params->res->data[0]);
      fflush(stdout);
#endif
      break;

      /*
       *  a slave is finished
       */
    case ExchFinished:
      params->curSlaves -= 1;
      break;

      /*
       * unrecognized exchange
       */
    default:
      ABORT( status, APPLYSEARCHH_EUEXT, APPLYSEARCHH_MSGEUEXT );
      break;
    }

    /*
     * Set the fraction of work remaining to be done, in this case the
     * fraction of slaves which haven’t yet finished
     */
    output->fracRemaining = ((REAL4)params->curSlaves)/params->numSlaves;
    if (params->curSlaves == 0)
    {
      /*
       * Count the number of 3-sigma samples - if more than 5%,
       * create an output, otherwise don't bother
       */
      const REAL4 std3 = 3.0*sqrt(params->var);
      const INT4 fivepct = 0.05*params->vec->length;
      UINT4 i = 0;
      UINT4 cnt = 0;

      for (i = 0; i < params->vec->length; ++i)
      {
        if (fabs(params->vec->data[i]) > std3)
        {
          cnt++;
        }
      }

      if (cnt > fivepct)
      {
        BuildOutput(status->statusPtr, params, output);
      }
    }
  }
  else /* no slaves left - finish up */
  {
    output->fracRemaining = 0;
    output->notFinished   = 0;
  }

  DETATCHSTATUSPTR (status);
  RETURN(status);
}

/*
 * Function ApplySearchSlave
 *
 * Called by LALApplySearch only on slave nodes. Each slave node
 * calculates one statistical moment. It decides which moment to
 * calculate based on it's rank. When done it transmits the result
 * to the master, then notifies the master that it's done.
 *
 * The database only has spaces for the first four moments (mean,
 * variance, skewness, kurtosis). Higher order moments are calculated
 * but otherwise ignored. If fewer than 4 slaves are started, the other
 * moments are set to a default value of 0.
 */
static
void
ApplySearchSlave(
    LALStatus*             const status,
    LALSearchOutput*       const output,
    const LALSearchInput* const input,
    HWSearchParams*       const params,
    MPI_Comm                    comm)
{
  /*
   * The moment to calculate, determined by node number:
   *   1 - mean
   *   2 - variance
   *   3 - skewness
   *   4 - kurtosis
   *   ... and higher order moments
   */
  const UINT4 moment_order = params->rank;

  MPIMessage mpiMsg;

  INITSTATUS( status, "ApplySearchSlave", MOMENTSC );
  ATTATCHSTATUSPTR (status);

  /* Initialisation */

  /* only do this part the first time the slave is called */
  if (params->initialised == 0)
  {
    snprintf(params->id, IDLEN, "slave [node %2d]", params->rank);

    params->initialised = 1;
  }

  /* Choose a different statistic to calculate based on the moment selected */
  switch(moment_order)
  {
  case 0:
    /* Should not occur, only the master has rank 0 */
    ABORT( status, APPLYSEARCHH_EUEXT, APPLYSEARCHH_MSGEUEXT );
    break;
  case 1:
    mpiMsg.msg = ExchMean;
    break;
  case 2:
    mpiMsg.msg = ExchVar;
    break;
  case 3:
    mpiMsg.msg = ExchSkew;
    break;
  case 4:
    mpiMsg.msg = ExchKurt;
    break;

    /* Higher order moments */
  default:
    mpiMsg.msg = ExchHOM;
    break;
  }
  mpiMsg.source = params->rank;
  mpiMsg.send = 1; /* 1 object to send */

  /*
   * Do real work somewhere here. Note that we should break our job
   * up into 10 calls and notify the master after each 1/10th
   */

#ifdef USE_LDASCAMPMOMENT
  /* Calculate the moment */
  LALLDASCampMoment(status->statusPtr, &(params->res->data[0]),
                    params->vec, moment_order);
#endif

  /*
   * Transmit to master (node 0)
   */

  LALMPISendMsg(status->statusPtr, &mpiMsg, 0, comm);
  CHECKSTATUSPTR (status);

  LALMPISendREAL4Vector(status->statusPtr, params->res, 0, comm);
  CHECKSTATUSPTR (status);

  /*
   * Notify master we're done
   */

  mpiMsg.msg = ExchFinished;

  LALMPISendMsg(status->statusPtr, &mpiMsg, 0, comm);
  CHECKSTATUSPTR (status);

  output->numOutput     = 0; /* no outputs */
  output->result        = 0; /* pointer to results */
  output->fracRemaining = 0; /* ignored in a slave */
  output->notFinished   = 0; /* tells the wrapper master this slave is done */

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

E moments.schema

```
#
# lam boot schema for moments shared object
#
h -np 8 wrapperAPI -mpiAPI="(marfik.ligo.caltech.edu,10000)" -nodelist="(1-7)"
-dynlib="/home/charlton/local/lib/lalwrapper/libldasmoments.so"
-dataAPI="(datahost,5678)" -resultAPI"=(reshost, 9101)" -filterparams="(0,0)"
-realTimeRatio=0.9 -doLoadBalance=FALSE -dataDistributor=W -jobID=8
-uniqueID=9.0 -inputFile="/home/charlton/local/share/lalwrapper/wrapper.ilwd"
```