

Report of the Data Analysis Software Working Group (DASWG) Sub-Committee on "What next after LDAS 1.0?"

v.4, March 9, 2004

Kent Blackburn, Patrick Brady, Erik Katsavounidis, Alan Weinstein (chair)

The sub-committee was asked to address the question of what should happen to LDAS development after the release of LDAS 1.0 (which happened on 2/14/04), and to consider the "charges" listed below. The sub-committee exchanged emails, met once on Feb 27, and reviewed a document prepared by Tom Nash (at the request of Albert Lazzarini) on "LIGO Computing Environments and Analysis Quality Assurance", LIGO-L030155.

Charge 1) Accessibility of LDAS to the LSC. Please consider, among other things, ease of use.

The committee interprets this charge to mean that we should gather information on who uses LDAS, what they use it for, whether they find it easy to use, what their plans are for future use, what features of LDAS they find to be essential, and what additional features they would like (or need) to see.

LDAS logs all users' requests, and the CMON utility provides reports on LDAS use. It reports that several dozen members of the LSC have used LDAS in the last few months, and over 5 million jobs have been submitted to the production clusters since S1. LDAS is used for many different tasks:

- Short data or metadata queries using GUILD or LIGOTools.
- Metadata database insertion or extraction.
- Acquiring frame data, delivery as frame files or LIGO LW data files.
- Conditioning frame data prior to delivery.
- Generation of reduced datasets via createRDS for LSC-wide use.
- Creation of custom RDS's for specific uses.
- Providing location of online data for online processing.
- Analyzing data with LAL-based GW search software via LALWrapper DSOs.
- Generate custom data products necessary for the continuing development of DSOs, including new DSO search codes.

In the context of the DASWG, information on the last two task categories may be of greatest interest, although we do not mean to underestimate the importance of the other task categories, such as the createRDS facility or the use of the database by online DetChar software. The subcommittee is aware of significant use of LDAS by the Burst group (Tfclusters, Waveburst DSOs) as well as some evidence of significant use by members of the Stochastic, Pulsar, and Inspiral groups. The subcommittee decided that the best approach to addressing this charge would be to distribute a short questionnaire to the entire LSC, and compiling the results. This questionnaire is appended to this report, followed by a brief summary of results.

Charge 2) Platform and other support of LDAS.

Charge 3) Personnel requirements if LDAS goes into maintenance mode with the release of LDAS 1.0. Please compare to requirements for development of LDAS 1.0 or continued development of all components of LDAS.

These charges were combined because much of the maintenance requirements for LDAS depend on which platforms are supported. Specifically, several major proposed development tasks for the LDAS team involve extending support for different compilers and platform architectures. LDAS is built on top of a very large collection of third-party software packages collectively referred to as LDCG. As directed by LIGO Lab and the LSC, both LDCG and LDAS software proper are currently supported on Sun Solaris 9 and Intel RedHat Linux 9; on both platforms, the GCC 3.3.1 compiler is used.

Following the release of LDAS v. 1.0.0, Kent has compiled a long list of potential future development projects. We summarize them here without any supporting details; just ask. All estimates of time required to implement these goals are very approximate.

- Port to the Solaris C++ compiler - 2-4 months for one programmer.
- Port to the Intel C++ compiler - 1-2 months for one programmer.
- Implement fully federated LDAS database - 4-6 months for 1.5 people.
- TCL/Globus software development - 4-6 months of a new hire to begin real work on this long-term project.
- Maintenance of underlying tool sets (LDCG) - 1-2 FTE's, ongoing.
- Further development of data conditioning tools - 1-2 FTE's, ongoing.
- Maintenance of LDAS proper, including modernizing socket communication technology, and more - at least 2 experienced FTE's, ongoing.
- Port to 64bit INTEL or AMD CPUs - 1-2 FTE's for 6 months.
- RPM based distributions of LDAS - 1 FTE for 6-9 months.
- Multi-FrameAPI topology - group effort, 6-9 months.

The lab-based LDAS team maintains 6 clusters (LDAS-WA, LA, CIT, TEST, DEV, MIT) which run LDAS. There are similar linux-based beowulf clusters maintained at and by UWM, PSU, AEI, and UTB, although some of these may not be currently running LDAS (LDAS was designed as a software environment primarily for the lab-based clusters). The linux-based beowulf clusters can be configured to run condor; currently, only LDAS-CIT amongst the lab-based clusters is so configured. However, the front-end servers that LDAS APIs run on are specialized hardware (Sun Servers, Multi-CPU boxes that are not supported or practical for a condor environment. They would most likely have little role outside of LDAS.

It should be noted that LDAS software development is tested using a very mature set of test suites that include nightly builds, nightly testing of LDAS against every single change in the code, a problem tracking system, etc. This type of QA infrastructure is in general (and was for LDAS) expensive to establish.

Charge 4) Possibility to extract pieces of LDAS 1.0 as components of a revised on-line analysis system.

LDAS currently provides certain services that are central to near-real-time data processing:

- Data discovery – LADAS maintains a table of locations of frame files produced by the online data acquisition system, and is made available for use by any job that inputs frame data, both near-real-time and off-line. This information is used by createRDS, and by near-real-time LDAS/LAL-based GW search software.
- The LDAS-based createRDS facility runs near-real-time, creating RDSs for distribution to other computer clusters for use by LSC data analysts.
- In the first three science runs, LADAS/LAL-based GW search software was run on the site LDAS clusters near-real-time, in an attempt to provide useful feedback to operators and scimons. There is no consensus that these efforts were particularly useful.

It is certainly possible to extract pieces of LDAS code. LDAS software is organized as stand-alone c++ libraries (such as the frame or datacond libraries), and code in those libraries can be called from TCL scripts; this is how most of LDAS is implemented. This code is available for use in other applications. Of course, revised systems for replacing the role of LDAS for the first two services listed above would require considerable effort.

With regard to the datacond library: this is also a stand-alone c++ library that can be used in other applications. It should be noted that LSC data analysts make use of at least four different extensive software libraries for data conditioning (signal processing): MATLAB, DMT, LAL, and LDAS/datacond. This is a lamentable duplication of effort, creates confusion, and makes validation of all software more laborious.

LDAS Questionnaire, March 2004

The LSC Data Analysis Software Working Group (DASWG) sub-committee on LDAS requests your input: do you use LDAS, what do you use it for, do you find it easy to use, what are your plans are for future use, what features of LDAS do you find to be essential, and what additional features would you like (or need) to see.

This is a short questionnaire; please respond in the next few days. Many thanks!

Q1: Do you use LDAS for any/all of the functions (Q1.1-6) listed below?
[] YES
[] NO

For each function (Q1.1-6), please specify your level of "Usage":
0=not at all, 1=light, 2=average, 3=heavy

For each function (Q1.1-6) that you use, please give a "rating":
0=don't know, 1=useful/functional, 2=needs work, 3=difficult to use

	Usage	Rating
Q1.1 Short data or metadata queries using GUILD or LIGOTools	[]	[]
Q1.2 Metadata database insertion or extraction	[]	[]
Q1.3 Acquiring frame data, delivery as frame files or LIGO LWfiles	[]	[]
Q1.4 Conditioning frame data prior to delivery	[]	[]
Q1.5 Creation of custom RDS's for specific uses	[]	[]
Q1.6 Analyzing data with LAL-based GW search software via LALWrapper DSOS	[]	[]

Q1.7 Which LAL-based GW search DSO(s) do you use under LDAS [.....]

Q2.1 Do you envision continuing the use of LDAS through the next 12 month?
[] Not at all, [] Lightly, [] Heavily, [] I'm phasing out

Q2.2 If you are moving to other tools/platforms, which are you considering:
[] I'm sticking with LDAS, [] Condor, [] MATLAB, [] DMT,
[] Other(specify)
What is your primary reason for considering another platform? [.....]

Q3: What is your opinion of the ease of use of LDAS, apart from issues of ease of use of LAL software?
0=don't know, 1=useful/functional, 2=needs work, 3=difficult to use []
Please add specific comments.

Q4.1 What features of LDAS do you find to be essential?

Q4.2 What additional features would you like (or need) to see?

Results from LDAS Questionnaire for DASWG, March 2004

The questionnaire was sent out Tues 3/2/04, 9am; almost all responses were received by Friday 3/5/04, 9am. 40 responses were received, including two that were on behalf of the ACIGA and PSU groups.

Q1: Do you use LDAS for any/all of the functions (Q1.1-6) listed below?

24 out of 40 responded YES. Of these, the number using the following features of LSAS were:

Q1.1 Short data or metadata queries using GUILD or LIGOTOOLS	22
Q1.2 Metadata database insertion or extraction	9
Q1.3 Acquiring frame data, delivery as frame files or LIGO LWfiles	18
Q1.4 Conditioning frame data prior to delivery	8
Q1.5 Creation of custom RDS's for specific uses	5
Q1.6 Analyzing data with LAL-based GW search software	9

Most users of the functions in Q1.1-Q1.5 rated them 1=useful/functional or 2=needs work.

Most users of the functions in Q1.6 rated them 3=difficult to use.

Q1.7 Which LAL-based GW search DSO(s) do you use under LDAS?

Q2.1 Do you envision continuing the use of LDAS through the next 12 month?
[] Not at all, [] Lightly, [] Heavily, [] I'm phasing out

All but 1 said that they had already switched to other environments like Condor (the users of *inspiral*, *power*, *stochastic*, *knownpulsardemod*, *stackslide* DSOs) or were planning to switch before S4 (the users of *BurstDSO*, *tfclusters*, *waveburst*, *exttrig*). One user (of *hierarchical inspiral*) intends to continue use of LDAS to run *dataconditioning* and the search DSO.

Some comments received, in random order, were:

- *knownpulsardemod* and *stackslide* -> I am no longer running DSOs; I've switched to Condor. Running DSOs was always difficult. The functionality I needed to do I/O of SFTs with LDAS was not fully implemented until September of 2003. By that time it was already becoming clear that Condor was the way to go. The *knownpulsardemod* and *stackslide* DSOs are obsolete. I am not using them. In the unlikely event that LDAS is used for online analysis to provide figures of merit for the control room, they could become useful again. However, I think the DMT will handle that. I have already phased out running DSOs. I will be running LDAS *createrds* jobs, heavily of course during runs. Using Condor it has been much easier to develop code, validate code, run code in parallel, and process results. It is also the only way I can imagine easily running Monte Carlo simulations. It also seems faster than LDAS.
- R - www.r-project.org
- *hierarchical inspiral* and *Datacond*. I am sticking with LDAS because I have spent quite a lot of time in developing the DSO on LDAS. Now, I would like to run the DSO on real data. At this stage, I cannot shift to another platform.
- Have used *FCT* DSO, but not for a while. I use all of the above tools for various activities. As indicated above, I primarily use LDAS for access to frame files and database, data reduction, data conditioning. For data analysis I mainly use Matlab and C/C++ programs when prototyping. In the past I have usually made standalone codes/libraries but also incorporated those codes into LDAS, either in the *dataconAPI* or as a search DSO.

- At this time none. Block Normal was originally coded as a LAL-based DSO, based on prototypes written in Matlab. The overhead and time delays in development associated with having to work with not one but two (LDAS and LAL) environments that were themselves under development and on different schedules ultimately led to our ending efforts to port the Block Normal code
- inspiral -> Condor. Speed of completing analysis. Also:
 1. Better Job management: Condor DAGman make life much easier when trying to manage analyses.
 2. Ease of developing and debugging standalone code vs developing under LDAS.
 3. Ease of access to data and data products.
 4. Rapid turnaround of bug fixes.
 5. Reliability.

Q2.2 If you are moving to other tools/platforms, which are you considering:
 I'm sticking with LDAS, Condor, MATLAB, DMT,
 Other(specify)

All but one LDAS user checked Condor, Matlab, and/or DMT. One (using hierarchical inspiral) said he's sticking with LDAS. One user said he uses R.

Q3: What is your opinion of the ease of use of LDAS, apart from issues of ease of use of LAL software?

Some comments received, in random order, were:

- Learning curve is very steep
- LDAS is inflexible, failure prone, has no easy way to string together jobs into a full pipeline, and terribly inefficient.
- LDAS is too bulky and it reminds me of a suit of corporate software they use in big companies, banks. I do not think that LDAS's model is appropriate for the fast moving LIGO research environment.
- LDAS is difficult to use, based on what I've read about it and have heard from experienced users. As a non-user, I have been struck by the cumbersomeness of communication with the DSO. I like to have a fair amount of autonomy when writing code, including easy file reading/writing. I understand that there have been improvements in this area recently, but I haven't followed the issue closely.
- LDAS is stuck with LAL/LALWrapper and I find LAL/LALWrapper completely unacceptable and highly unproductive framework (wrong language, horrible "standards" and software design decisions), I'd like an environment that would allow me to write a program in the language or combination of languages of my choice and not being constrained by unusable standards; Condor seems to be the right step in this direction since it is very flexible and does not depend on the language;
- I want to be able to develop a code that is fully functional on a single computer and then move it to the cluster environment without changing the code, just adding some scripting. Again that's very difficult to do in the existing LDAS/LALWrapper/LAL framework, you have to program from the beginning for LDAS/LALWrapper. Standalone LALWrapper programs are extremely inconvenient to work with.

- Within the existing framework there is a huge amount of totally unnecessary overhead involved even if one wants to do a simplest thing with the data. Many people periodically come to me with some data analysis ideas that they want to try but once they study LAL/LALWrapper/LDAS documentation and realize how many meaningless rituals they have to perform in order to multiply 2 by 2 in the existing framework, they disappear and I never see them again. A lot of good ideas were probably lost this way.
- It is very difficult to debug a program in the existing framework. There are too many things that can go wrong in LDAS/LAL/LALWrapper. The whole system is way too complicated and I do not think all this overhead is really worth the effort. I would prefer a simpler system. Again, I think Condor is the right direction to move in that respect as well.
- It is very difficult to quickly test various ideas in the existing framework because of the huge overhead.
- The existing framework (mostly LAL/LALWrapper) introduces totally unnecessary and artificial interdependence of various search codes. Waveburst, for example, does not use anything from the rest of LAL but nevertheless I have to spend hours compiling the code for all the other ETGs to be able to work on my code. If somebody committed bad code to CVS, all other ETGs suffer. Developers should be able to choose what packages they need to build besides their own.
- The problem is that you cannot really separate LDAS from LAL for serious data analysis tasks. One can only do auxiliary things using purely LDAS. For example, I use LDAS' datacon for MDC frame production. RDS frames are also generated using purely LDAS. One cannot do data analysis in LDAS without using LAL at the moment.
- Much of the documentation for raw frame queries or simple data-conditioning is hard to follow. Probably the only exception to that is the calibration routine, which gives a several-step example of how to calibrate your AS_Q data. More examples of simple queries would go a ways to make LDAS easier to use.
- The error messages from LDAS can be quite cryptic. For example, in developing a data-conditioning pipeline I've come across messages on the order of: "Empty data bucket" or "Received int; expecting int" (referring to long & short ints, I guess). Sometimes LDAS just hangs for no reason; was it my request, or is it just a bum job request?
- My use of LDAS is so minimal that I don't think I can judge its ease of use
- Errors are not easily debuggable without expert involvement
- I think the main things which have steered people away from LDAS are:
 - The difficulty of developing/debugging a DSO. Developing within a "standalone wrapperAPI" does work, but you have to prepare an ilwd file with the necessary input data, and it always gives the impression of being a hacked-together development paradigm -- even though it is really the only way to do it. The MPI framework has a learning curve which must be overcome. Some debugging requires running full-scale jobs within LDAS, and originally, there was no way at

all to get debugging information out. The situation is better now that running a DSO from the /dso-test directory is the norm, and now that it is permissible for a DSO to write to a disk file.

- The difficulty of getting job output. Dumping lots of events into the database leads to very long and painful retrieval times, due to the nature of the database (which generally has to sort through all the aggregated data to satisfy a query) and to the limitation on how much output LDAS can deliver in one job, necessitating repeated queries. Users are generating MANY more triggers than originally anticipated (and outputting all of them, rather than just filling histograms and outputting those), and the database architecture just does not scale very well.
- LDAS is very difficult to use - and involves a very tedious learning process if one is developing a DSO. I have barely managed to put my DSO together. However, I find the data conditioning in LDAS to be very useful. Ideally what I would want is the freedom to run parallel codes (written using LAL routines) on any cluster without strict rules of LDAS together with the advantage of getting the data preconditioned before my code touches it.
- Accessing LDAS via Guild, the Control/Monitor GUI or the RDS GUI is fairly painless. These tools are great for certain tasks (access to data, data retrieval, monitoring) but aren't designed to do analysis. Doing data analysis mostly requires writing long-ish scripts (usually in Tcl but any scripting language will work). I don't have any problems with that method, except that when scripting for LDAS the obscurity of many error messages makes it harder than it needs to be. If I write an LDAS command with a mistake in it, an error message will be generated but often the message is not very helpful.
- Reasons for having switched to MATLAB: Short turn around time for writing, testing, and debugging application code. Ease of verifying that linear algebraic intensive code matches the algorithms. Stability of the development environment. Faster execution of code, even in the interactive mode.
Reasons for considering Condor(and other job management software): Ease of use and the ability to run any executable.
- LDAS was very difficult to install, in part due to the large number of configuration files that needed to be tweaked for local use. Updating to newer versions of LDAS typically made the older configuration files obsolete, and thus requiring working with all of the new configuration files and again going through and changing a large number of parameters spread across many files.
- There are several features in LDAS that make using it difficult. Very tight restrictions on how data is passed between functions means that for certain applications (e.g. Base-banding) convoluted coding was needed to work around these limitations. Although ultimately possible to do this, it lead to much more obfuscated code that was therefore more difficult to debug and verify.
Writing events to the database could be very slow and therefore limited the ability to use the database for final outputs.
No ability to preserve state and load in a different job hampers the ability to process very large quantities of data as a single stream.
The job manager's communication with only one data-con server proved to be a major

bottleneck for massively parallel processing of data offline, effectively forcing the use of data-conditioning within our DSO so that the computational burden could be distributed. This is undoubtedly a result of the unforeseen fact that the demands of the different DSO's leads to them each handling the data-conditioning step differently.

Q4.1 What features of LDAS do you find to be essential?

Some comments received, in random order, were:

- Being able to get data and on-line search results.
- tape archive of full frame data, trend frame data, QFS file systems.
- Originally, I thought we needed a common vetted set of software but I can also see the advantages of having multiple analysis methods for the same analysis problem as a means of checking and to allow flexibility. The problem, of course, is that one now needs an additional layer of organization to check that the software and analysis methods are valid. In the end that additional burden is most likely worth it. I expect that the new analysis and software committee will move into this job.
- access to the large data sets
- The createrds jobs are essential! The LDAS createrds jobs and the
- createrds scripts, along with the ligotools LDASjob package ran flawlessly during S3. This functionality must be preserved. LDAS should be congratulated for creating a very robust RDS generation process!
- Querying the trigger databases, imprint selected data, and choosing playground segments.
- Framecpp library until it is replaced with something faster is essential in the DAQ system. We might want to stick with it in the NDS where it works nicely.
- Creation of RDS works great. It could be done other ways, but I'm pretty happy with the present system.
- diskcacheAPI helps to find data (I do not know how LDR compares to that). dataconAPI has some useful functionality, in particular an ability to apply calibration. However, what I hate in dataconAPI is that it has its own rather limited language. It would be more usable if it were just an extension of TCL or something like that.
- Data archiving, retrieval, conditioning.
- Accessibility of frame data. What would be even better is to be able to combine frame elements (e.g. H1 & L1 AS_Q channels) from different frames without having to resort to a data-conditioning call (which says, 'output(H1,,H1:LSC_AS_Q,H1 gw channel); output(L1,,L1:LSC-AS_Q,L1 gw channel);'
- Science mode segments. Using the segwizard makes finding data-quality flags much easier. Unfortunately, this is probably more dependent on LIGOTools, although I expect the segments may be stored on LDAS metadatabase-- I don't know.
- Logging/tracking of results; code release model/validation/regression cycle
- we are able to work without LDAS now [stochastic].
- none, some like GUILD are occasionally handy
- LDAS makes a good data server, delivering the desired time interval and channels to remote users. It is the basis for remote data access using getFrames and using DTT, although some users of these tools might not be aware of this fact.
- createRDS
- Data conditioning.
- Retrieving data remotely (frame data, database information etc); Processing data remotely (ie. at the sites, but I also regard the CIT installations of LDAS as "remote" from me since it's not

a program running on my own desktop). The convenience here is that the (vast) amount of LIGO data is stored on a remote machine, but is visible to LDAS. I don't have to get that data locally and find storage for it in order to work on it; Data reduction; Data conditioning (as distinct from data analysis)

- The production and publication of RDS frames. Although these are not produced by anyone here using LDAS, We do use these reduced frame files extensively. We assume that the people who make these files are in the best position to decide the easiest mechanism to accomplish the production of these files and to date that has meant LDAS.

Q4.2 What additional features would you like (or need) to see?

Some comments received, in random order, were:

- Passwords that last indefinitely. Access to all systems by default.
- more disk capacity for frame builders.
- None. Although I consider RDS generation and (possibly) data discovery for online searches to be important.
- None for me personally. It would seem to me that if DSOs are phased out (which seems to be the trend) that a lot of LDAS code could be exposed as a library for C++ developers, maybe for running online searches using the DMT?
- Faster frame file creation in framecpp library I would like to see.
- Easy production of web pages with plots by search code running in near real-time at the sites.
- LDAS should not rely on LAL/LALWrapper and provide way to use other frameworks and languages, in particular arbitrary C++ code.
- Easier installation of LDAS at local institutions.

- * How about plug-and-play job scripts which perform some of the common functions, like calibrating data, combining channels from different sites, and outputting channel data in a format displayable by DTT? I know that these scripts exist, but I'm betting they haven't been made available because every new release of LDAS tweaks something which causes scripts to crash.

* How about returning a Matlab-compatible script with the data output (be it LIGO-LW or Frame data) which will extract out the various products from the file for quick viewing? Right now when I get a LIGO-LW file, I have to try a series of commands to try to find where the time-series is! I would really much rather use something like: `>> H2_asq = job1548_query('H2\LSC-AS_Q','TIME','data');` Your TIME domain data for H2\LSC-AS_Q is loaded, sir! `>> who H2_asq H2_asq: [1x1024 double]`

- Ability to spawn multiple instances of APIs, e.g., dataCond or FrameAPI would immediately relieve bottle necks (perceived and real).

- Unfortunately, the LDAS development/release cycle is so long that I no longer have any enthusiasm for suggesting new features. It is usually easier to just find a workaround.

- multiple instances of the frame and datacondAPI's to allow fast parallel I/O. In the current model, much of the work of these API's is moving into the search codes to get parallelism.

- I have no idea. I have just begun running my DSO in LDAS - and I have very little experience so far.

- * LDAS error messages need to be de-mystified (a minor issue is that LDAS messages tend to have some extraneous junk in them which is useful for developers but confuses everyone else. That could be cleaned up).

* I would like the dataconAPI command interpreter to be extended in certain ways. For example:

- allow standard math symbols like +, -, *, / to be used in the usual way
- support composition of functions such eg. "x = add(y, add(z, w))" for $x = y + z + w$

An obvious step after these two would be adding if-then statements and looping. This would just bring the interpreter into line with other simple languages like matlab or shell scripting and make it behave in a way which people are already familiar with.

- A single well commented configuration file for all of LDAS. Within this configuration file, parameters that are associated with the local computing environment should be located together.
- Intelligent managing of job output directories.
- Significant improvement in data access speeds.
- Ability to save/load state.
- Ability for the job manager to communicate with more than one data-con server to distribute the data-conditioning load.
- Flexible mechanism for passing data between functions

A more general set of comments from one respondent:

First let me say that LDAS did make changes to address some of the below (for example by allowing direct I/O from DSOs). Also, I do not think that the learning curve for developing a DSO is steeper than that for developing LAL code to run under Condor. However, the DSO development process is inherently less flexible and less transparent than developing LAL code to run under Condor. LDAS was designed to fulfill a rather focused set of goals. Thus, in a way this question is "rigged". Developing DSOs will always seem difficult in comparison to Condor development, unless LDAS becomes Condor. With those caveats, here is an attempt to explain the difficulties I experienced:

1) I had no control over the I/O of SFTs. There is a very long history here, that I won't go into. As I said above, it was not until September of 2003 that all the requirements were met for I/O of SFTs that would allow a preliminary periodic analysis to completely occur using LDAS. Even then it was not entirely clear how to do a followup Monto Carlo study using LDAS (though I'm not saying it could not be done).

2) The fact that the standalone wrapper did not run in exactly the same way as a DSO on the cluster made it slower to develop and validate code. The standalone wrapper required ilwd input, while the DSO requested frame input. However, if you asked the datacon to dump its data, you got xml, not ilwd output. Making fake data for testing was a multi-step process that was often very tedious and time consuming. This did become easier with the dataStandalone command, but was never completely straight forward. If the standalone code worked, but the DSO failed, finding the source of the bug and getting the bug fixed within LDAS was a slow process.

3) Other issues associated with I/O within LDAS caused difficulties. For example, all dataPipeline commands have to go through the dataconAPI, even if no conditioning is needed; all output had to go through the metadataAPI even if only xml output was wanted, creating bottlenecks. Another example is that I/O with ilwd, xml, or frames in LDAS requires a translation

from the structs in LAL or the frame library to structs that LDAS provides. If you set the value of a member of an LDAS output struct it would not always appear in the actual output where you expected it would. Another example is that the input of ephemeris data by LDAS could not be done using existing routines for this in the LAL support package. Instead, those routines had to be duplicated in LALWrapper to work with LDAS structs. In summary, the LAL code allows a more straight forward interface for frame and xml I/O than LDAS provided or allowed.