LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T1600055-v7 *LIGO* September 9, 2016

Real-time Code Generator (RCG)
Version 3.0 Release Notes

R. Bork, D. Barker, K. Thorne, J. Hanks

Distribution of this document:
LIGO Scientific Collaboration

This is an internal working note
of the LIGO Laboratory.

California Institute of Technology
LIGO Project

Massachusetts Institute of Technology
LIGO Project

LIGO Hanford Observatory

LIGO Livingston Observatory

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

1 Introduction

The purpose of this document is to:

- Describe RCG changes/enhancements as part of the upgrade from V2 to V3.
- Provide installation instructions for the new V3 release.

2 Release History

All code releases are found within the advLigoRTS area of the CDS SVN.

- branch-3.0: Initial release for testing only. (October, 2015)
- advLigoRTS-3.0 – March, 2016
- advLigoRTS-3.0.1 – April, 2016
- advLigoRTS-3.0.2 – May, 2016
- advLigoRTS-3.0.3 – June 2, 2016
- advLigoRTS-3.1 - August, 2016
- advLigoRTS-3.1.1 – September, 2016

3 References

Other CDS documents provide more detail on the implementation and installation

LIGO-T1500143 aLIGO DAQ Dual Data-Concentrator Installation

LIGO-T1600059 New DAQ Decimation Filters

LIGO-T1600066 New RCG Decimation Filters for 16 kHz Models

LIGO-T1500115 Real-Time Code Generator (RCG) SDF Software

LIGO-T1500227 aLIGO DAQ hardware, software setup

LIGO-T1500458 Adding Ubuntu OS to existing aLIGO DAQ systems

4 Overview

This document provides update information for the core CDS RCG and Data Acquisition (DAQ) system software for RCG V3.x. It also provides detailed information on how to install this version of software. This information is provided in the following sections:

- Section 5: Documentation on new features added in this release.
- Section 6: Listing of all bugs fixed in this release.
- Section 7: Installation of ldas-tools upgrade
- Section 8: Large IFO system installation instructions.
- Section 9: Standalone system installation instructions
- Section 10: Workstation update instructions

5 New Features

New features are typically added to the RCG for two reasons:

- A formal Engineering Change Request (ECR). An approved ECR must be submitted to have changes made to the RCG beyond the fixing of code bugs, as submitted to the CDS Bugzilla. Changes made to the RCG V3.0 in response to two approved ECRs are described in the following section 5.1.
- In order to fix certain code bugs that have been reported, new features may be needed to support the fix. New features that resulted from bug fixes are described in section 5.2.

5.1 New Features - Release V3.0

5.1.1 Data Acquisition (DAQ) System Support for 64bit Double Type Data

Previous versions of the RCG only supported the acquisition of floating point data as 32-bit single precision. This release now supports acquisition of 64-bit double precision type data.

To define a channel to be saved as a double, the DAQ Channels part is used, as shown in the example below. The keyword 'double' is simply added after the channel name.

```
#DAQ Channels

M1OUT_TP0
M1OUT_TP1
M1OUT_TP2
M1OUT_TP3
M1OUT_TP4
INT32_TST_1 uint32
INT32_TST_2 uint32 256
ADC_FILTER_1_OUT double
ADC_FILTER_2_OUT double
ADC_FILTER_3_OUT
ADC_FILTER_4_OUT
ADC_FILTER_5_OUT
RMM_OUT_FILT_1_OUT
RMM_OUT_FILT_2_OUT
RMM_OUT_FILT_3_OUT
```

5.1.2 DAQ System Support for Redundant Data Paths

This release supports the use of two independent DAQ chains. This allows for installation of a second DAQ data concentrator feeding separate frame-builders, NDS servers, and GDS DMT computers. While this feature is included in this release, by default the compile option to do this is NOT set.

To accommodate a parallel DAQ data path, several changes had to be made to the system:

- FE computers must transmit their data twice, once to each data concentrator. The result is double the data traffic on the DAQ network.
- Separate EPICS Data Collection Unit (EDCU) software had to be developed to gather data from auxiliary (non-real-time) systems. This task used to be taken care of by the data concentrator. However, since data here is asynchronous, two independent data concentrators would most likely not end up with the same EPICS data at the DAQ time

epoch. This would result in frame writers connected to the two data concentrators writing non-identical data frames. To resolve this, separate EDCU code gathers these auxiliary system EPICS channels and then feeds this identical data stream to both data concentrators. This data feed uses the same synchronous EPICS data transfer routines used by real-time FE systems. This software runs in user space, so can be run on most any computer that has a DAQ network connection.

This is all detailed in a stand-alone document LIGO-T1500143 .

5.1.3 New Decimation Filters

Bugzilla 956: Waiting definition from Peter.

- The updated DAQ filters are described in LIGO-T1600059 .

5.1.4 Added Model Parsing Diagnostics

Bugzilla reports 895 and 966 were, in part, a request for more compile time checks, with better diagnostics and cleaner compile error messages. As a start in this direction, the RCG MATLAB model parser had some fairly extensive changes made for this release.

At compile time, the RCG will now produce four compilation diagnostic files in the build area. Two of these files are the same as previous releases:

- *modelname.log*: List of all of the steps taken to compile the model.
- *modelname_error.log*: Listing of any errors encountered in the compilation process.

The two additional files produced by this release are:

- *modelname_partConnectionList.txt*: Lists all of the parts from the model and their associated input and output connections. In previous releases, a single diags.txt file was written with this information, but was common for every compile and therefore overwritten and information lost when the next model was compiled.
- *modelname_partConnectErrors*: Listing of all parts which do not have their inputs properly connected in the MATLAB model.

Verifying that all parts have proper input connections is now done earlier in the build process to provide better error messages. As an example, the RCG error message output when building a user model with one part missing and input connection compiled with previous RCG versions is:

```
Parsing the model x1atstim16...
List of parts not counted:
ADC0 Adc
ADC1 Adc
TIM16_T3_GPST Gps
TIM16_BIO_ENCODE_BIT2WORD2_cdsBit2Word1 Bit2Word
TIM16_BIO_ENCODE_BIT2WORD2_Sum SUM
TIM16_BIO_ENCODE_BIT2WORD2_Gain Gain
TIM16_BIO_ENCODE_DIO_0_OUT EpicsOut
TIM16_BIO_ENCODE_SUS_HWWD_RESET EpicsOut
Please check the model for missing links around these parts.
make[1]: *** [x1atstim16] Error 1
make: *** [x1atstim16] Error 1
```

As can be seen above, this error message simply points the user to an area of the model to search for input connection errors.

Compiling the same model with this code release produces:

```
***ERROR: Found total of ** 1 ** parts with one or more inputs not connected.
See x1atstim16_partConnectErrors.log file for details.

A complete list of model part connections can be found in
x1atstim16_partConnectionList.txt
```

Note the new error message indicates how many parts have missing inputs and points the user to the partConnectErrors.log for the specific part information. For this particular example, the error file contains the following:

```
***
EpicsOut with name TIM16_BIO_ENCODE_SUS_HWWD_RESET has no input connected.
```

Bugzilla 966 expanded on this input checking to include user provided C code functions ie does the number of inputs/outputs connected to a C code block in the MATLAB model correspond to the number of inputs/outputs actually used in the user C code.

In order to perform this checking, the RCG uses a two step process:

1) An attempt is made at parsing the user C code to determine the number of inputs/outputs used in the code. This is fairly straight forward when the code is written with the I/O variables as array elements. For example, x = input[0], y = input[1]. These are then compared to the number of I/O connections in the user model.

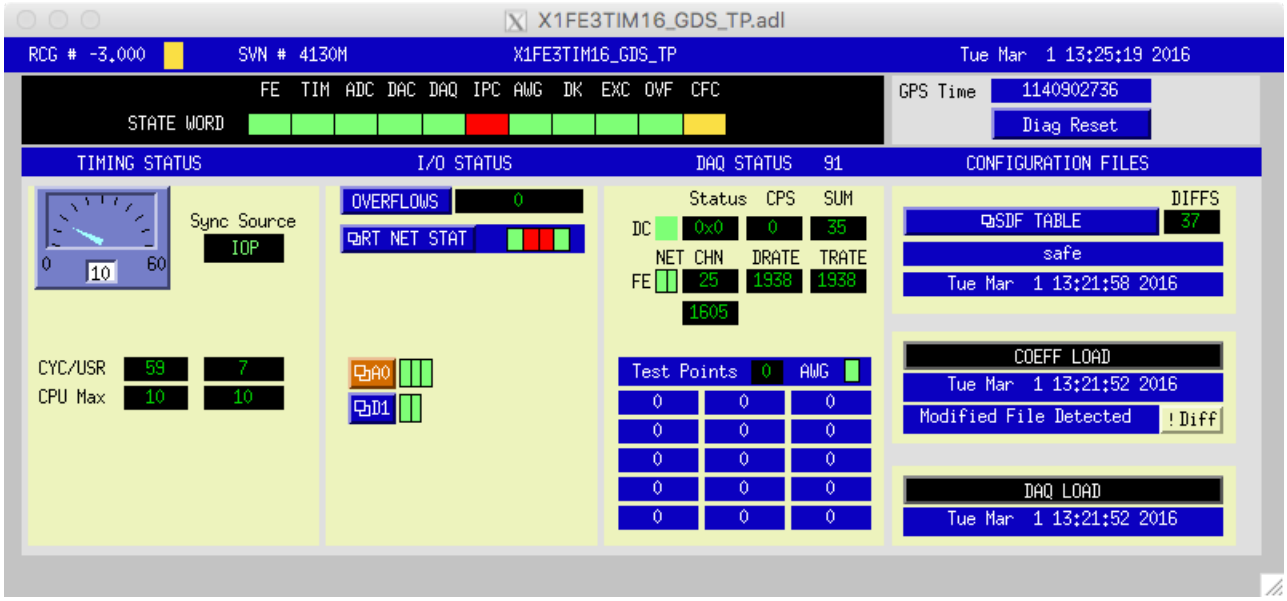
Many of the present user provided C code modules are written in this style, but a number are much more clever, i.e. use pointer increments and array increments in loops, adapt themselves to the actual number of inputs/outputs specified by the user model, etc. Determining I/O use in these C code files will require more intelligence in the RCG parser, for which there aren't resources to work at the present time. Therefore, to cover the models that the RCG presently cannot resolve, a second step is followed:

2) The RCG refers to the {RCG_RELEASE}/src/include/ccodeio.h file. This file contains user C code information as 4 columns per row:
   - Name of the C code file.
   - Name of the C code function.
   - Number of inputs
   - Number of outputs

5.1.5 GDS_TP MEDM Screen Additions

The auto-generated GDS_TP screens have changed slightly in this release, with a few additions:

- The RCG code version used to build the code is displayed in the upper left hand corner, in this case, -3.000. The minus (-) sign, plus the yellow block next to the release number, are a warning that this specific model was built from a non-released version of the RCG. For production systems built against code releases, these should not appear.
- COEFF LOAD area: An additional text field and "!Diff" action button has been added to provide additional information when filter coefficient files have changed from that presently loaded into the runtime FE code. More information on these charges are provided in the following subsection of this document.



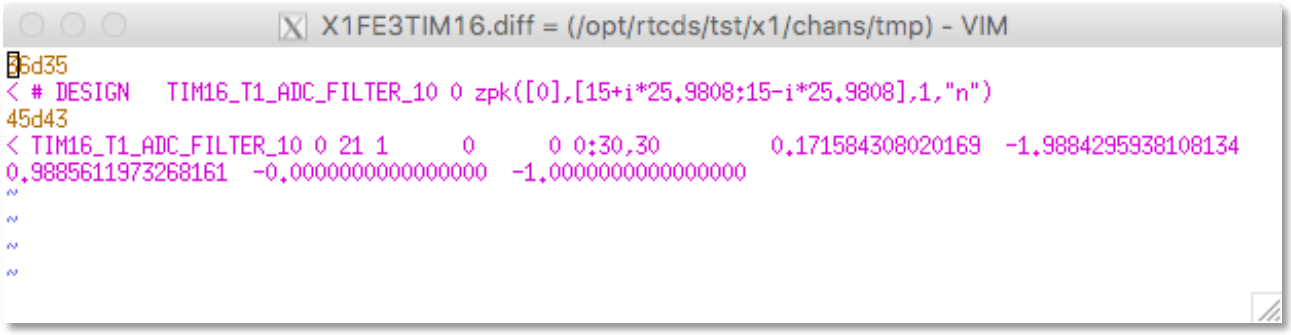
5.1.6 Filter Module Coefficient Loading

In response to CDS Bugzilla reports 894 and 896, the way filter coefficient files are handled has been changed. Previously, the GDS_TP screen would simply display "Modified File Detected", with no method provided to determine what was modified. Also, if individual filter module coefficients were loaded instead of loading all filters at once, the history of changes would be lost in the filter coefficient file archives.

To try and correct these deficiencies, this RCG release includes a number of changes:

- A new directory (tmp) is required below the filter module coefficient file directory eg /opt/rtcds/lho/h1/chans/tmp. On FE code startup, it will copy the associated filter coefficient (foton) file from the 'chans' directory into the 'chans/tmp' directory and then load coefficients from that file.
- Foton users should continue to define filters in the coefficient files in the 'chans' directory as the 'chans/tmp' directory is only for use by the FE code.

- Once a user saves his/her updates to the filter coefficient file (typically using the CDS 'Foton' program), a "Modified File Detected" message will appear in the COEFF LOAD section of the GDS_TP screen. The date and time that the FE last loaded a filter coefficient file will continue to appear on the screen as well. (See example GDS_TP screen in previous section.)
- If the user wants to see the difference between what is presently loaded into the FE code and the modified file, the '!Diff' button will bring up an 'xterm' windows with the list of differences. An example is shown below. This makes use of the Linux standard 'diff' function. Left and right chevrons (<, >) indicate which lines differ between the two files. A left chevron (<) indicates a line in the user's filter coefficient file. A right chevron (>) indicates a line in the FE code loaded file. In this example, two lines have been added in the filter coefficient file, indicated by '<', showing a single new filter that did not exist in the previously loaded file.



- At this point, the user has two options for loading coefficients from the new filter coefficient file:
  1) Load all changes at once by selecting the 'COEFF LOAD' icon on the GDS_TP screen. This causes:
     a) Filter coefficient file presently in the 'chans/tmp' directory, which had been used to load the present coefficient settings, is copied to the 'filter_archive' area.
     b) The new filter coefficient file is copied into 'chans/tmp' area.
     c) FE code reads in all coefficients from file in 'chans/tmp' area.
     d) On the GDS_TP screen:
        i) "Modified File Detected" message will disappear.
        ii) Date/time of coefficient load will update to present date/time.
  2) Reload coefficients for an individual filter module using the "LOAD COEFFS" icon on the filter module MEDM screen. In this case, the sequence is slightly different:
     a) Filter coefficient file presently in the 'chans/tmp' directory, which had been used to load the present coefficient settings, is copied to the 'filter_archive' area.
     b) Software new to this release then:
        i) Parses the new filter coefficient file, extracting only coefficient information associated with the particular filter module that is to be updated.
        ii) Parses the existing filter coefficient file file in the 'chans/tmp' directory, extracts the present data for that particular filter module and replaces it with the data from the new filter coefficient file.
     c) FE code reads in coefficients, from the 'chans/tmp' area file, for that particular filter module.

d) GDS_TP screen gets updated.
   i) 'COEFF LOAD' date/time is updated to present date/time.
   ii) If the filter coefficient file in the 'chans' directory now matches the filter coefficient file in the 'chans/tmp' directory, then the "Modified File Detected" message will disappear. This would be the case where only coefficients for the individual filter module just loaded had been changed in the new filter coefficient file.
   iii) If there are still differences between filter coefficient file in 'chans' and 'chans/tmp', then the "Modified File Detected" message will remain.

5.1.7 Setpoint Monitoring

Some new features have been added to setpoint monitoring (commonly referred to as SDF) as the result of fixes to reported bugs. Of particular note:

- Monitoring of filter module switches can now be set on an individual switch basis. In the past, monitoring was set on an all or nothing basis.
- The various SDF table views have been made more consistent. The actions of revert, accept and monitor can now be done on any of the table views.
- High precision values can be saved in the system.
- A EPICS Channel Access build is available, initially targeted to monitor the Beckhoff control systems without modifying the Beckhoff IOC.

Separate SDF documentation (LIGO-T1500115) is in the process of being updated to reflect all of the recent changes.

5.1.8 IOP Startup Prevention

As noted in bug report #903, the startup of the IOP with faulty (non-responsive) PCIe I/O cards in the I/O chassis can lead to bad results. For example, if an IOP model is built to use 4 DAC cards and the third DAC card does not respond to PCIe initialization, the IOP will actually report the fourth DAC as bad and start routing data intended for the third DAC to the fourth DAC. On startup, the IOP scans the PCIe bus and has no way of determining exactly which card is faulty and simply reports that, for this example, only 3 DAC modules were found and fourth is missing.

To prevent this situation, particularly on production systems:

- Add the line 'requireIOcnt = 1' to the Parameter block in the IOP model.
- In response to this flag, the RCG will generate a startup script that includes checking of the PCIe bus for the appropriate number of modules prior to starting the IOP code. This new startup script makes use of the Linux 'lspci' command to check that all of the required modules are present. If not, the script will exit with an error message.

5.2 New Features – Release V3.1

Release V3.1 was developed to reduce overall DAQ network traffic at the LIGO sites, in accordance with ECR LIGO-E1600234. Analysis of DAQ network traffic showed that data from IOP models was running about 15MByte/sec. This was not data to be actually recorded by the DAQ system, but was generated simply by the RCG requirement that all control models have a minimum of 2 DAQ channels. With IOP models running at 64K samples/sec, this produced

~500KB/sec per IOP, with LIGO sites having ~30 such processes. This puts an added burden on the DAQ network, along with additional computation time on the DAQ computers for CRC calculations and checking. This was not a particular issue in the past, but as the DAQ network traffic has continued to expand due to addition of more DAQ channels, site DAQ traffic has gone to the 70MB/sec range. More DAQ channels are planned to be added in the future for O2.

Therefore, the RCG code was modified for this release to not require that a control model have any DAQ channels. A side effect of this change is that the DAQ data latency is reduced by 1/16 of a second, as the real-time component no longer has to double buffer data, done previously due to computational time constraints.

6 Bug Fixes

A complete listing of bugs fixed in this release is provided in the following sections.

6.1 Base Release V3.0

Bugzilla Number	Description	Comments
494	Remove test for frame directory count when using GPS time folders	
691	DAQ build should detect EPICS, framecpp installs	
831	Add support to acquire data as 64bit, double precision floating point format	New feature added per ECR. See section 5.1.1.
886	Add RCG version number to runtime code	Version number appears at top left of new GDS TP screen and recorded in frames as an EPICS channel. See section 5.1.5.
887	SDF uninitialized table does not accept present settings	
889	SDF should provide for monitoring of individual bits in filter module switch settings.	
894	Potential errors in archived filter coeff files.	See section 5.1.6
895	At end of compile, point user to actual log files.	See section 5.2.2
896	Generate complete list of modified filter banks instead of just 'Modified Filter File Detected'	See section 5.1.6
903	Add option to prevent IOP task start if not all IOP model defined I/O modules are present on the bus.	See section 5.1.4
917	SDF table does not allow accept and mon settings in the channels not initialized view.	
918	SDF revert and accept do not work in the SDF full table view.	
919	SDF table should indicate when Sort on Substring is selected.	
921	SDF table should indicate when a channel represents a filter module.	

940	SDF does not handle very large values.	
948	Bad entries in ADC channel listings	
951	Precision errors in SDF monitoring	
966	RCG should verify user C code input/output parameters	See section 5.1.4
968	Update RCG to support Dolphin DX release 4.4.5.	
973	Update RCG SUS hardware watchdog (HWWD) monitoring part to handle status bit inversion of latest revision of the HWWD chassis.	
983	Update daqd, nds to build against ldas-tools 2.x	

6.2 Release 3.0.1

Bugzilla Number	Description	Comments
	Fix to startup phased of phase locked oscillator parts.	
	Fix to GPS part: time being passed to part was 1 cycle late.	
832	Dataviewer –r (restore from file) command line fixed.	
999	Correctly handle Killed real-time code in start script	
1000	Use code path variables in ccodeio.h	

6.3 Release 3.0.2

Bugzilla Number	Description	Comments
	Bug fix for SUM part connection checking.	
	Bug fix for autogeneration of ADC monitor screens.	

6.4 Release 3.0.3

Bugzilla Number	Description	Comments
1008	Coefficient Diff files are constantly updated after an individual filter coeff update.	
1014	Error in loading individual filter coefficients when name of one filter is contained in the name (substring) of another filter.	

6.5 Release 3.1.1

Bugzilla Number	Description	Comments
1030	Ramp Mux Matrix settings will not load on code startup	Fixed by setting LOAD_MATRIX to one on code initialization

7 Installation of ldas-tools upgrade for DAQ

7.1.1 Install ldas-tools 2.4.2 for Gentoo DAQ

The DAQ software now works with the newer ldas-tools builds (which contain framecpp). These also have the latest leap-second info. Here we download and install a pre-built package.

From the DAQ software website, download the pre-built ldas-tools-2.4.2-gentoo.tar.gz

1. Log on as 'controls' to that NDS server
2. Copy the downloaded file to /opt/rtapps/tarfiles
3. cd /opt/rtapps
4. tar –xzf tarfiles/ldas-tools-2.4.2-gentoo.tar.gz
5. rm ldas-tools
6. rm framecpp
7. ln –s ldas-tools-2.4.2 ldas-tools
8. ln –s ldas-tools-2.4.2 framecpp

7.1.2 Update ldconfig on Gentoo DAQ computers

On each Gentoo DAQ computer, we need to update the file telling the system where the framecpp library files are, as the new build does not have an extraneous 'linux-x86_64' sub-directory.

1. Login on to each Gentoo DAQ computer as 'root'
2. cd /etc/ld.so.conf.d
3. Edit framecpp-linux-x86_64 file to have the following

```
/opt/rtapps/framecpp/lib
```

4. ldconfig (to update LD paths to point to new ldas-tools areas)

7.1.3 Install ldas-tools 2.4.2 for Ubuntu DAQ

The DAQ software now works with the newer ldas-tools builds (which contain framecpp). These also have the latest leap-second info. Here we download and install a pre-built package.

From the DAQ software website, download the pre-built ldas-tools-2.4.2-rtubuntu12.tar.gz

1. Log on as 'controls' to the secondary frame-writer
2. Copy the downloaded file to /opt/rtapps/ubuntu12/tarfiles
3. cd /opt/rtapps/ubuntu12
4. tar –xzf tarfiles/ldas-tools-2.4.2-rtubuntu12.tar.gz
5. rm ldas-tools
6. rm framecpp
7. ln –s ldas-tools-2.4.2 ldas-tools
8. ln –s ldas-tools-2.4.2 framecpp

7.1.4 Update ldconfig on Ubuntu DAQ computers

On each Ubuntu DAQ computer, we need to update the file telling the system where the framecpp library files are, as they have moved. The existing 'framecpp-linux-x86_64' file does not need to be updated

1. Login on to each Ubuntu DAQ computer as 'cdsadmin'
2. 'sudo ldconfig'

7.1.5 Build ldas-tools-2.4.2 from source (optional)

If you wish to build ldas-tools 2.4.2 from source, do the following

From the DAQ software website, download the source code tar-ball ldas-tools-2.4.2.tar.gz .This can also be obtained from LIGO software site at ldas-tools-2.4.2.tar.gz

1. Log on as 'controls' to that NDS server
2. Copy the downloaded file to a build area (i.e. /opt/rtcds/projects/ldas-tools)
3. cd /opt/rtcds/projects/ldas-tools
4. tar –xzf ldas-tools-2.4.2.tar.gz
5. cd ldas-tools-2.4.2
6. Now configure it

```
./configure --prefix=/opt/rtapps/ldas-tools-2.4.2 --disable-tcl --enable-64bitrm framecpp
```

7. make
8. make install
9. cd /opt/rtapps
10. mkdir ldas-tools-2.4.2/etc
11. cp ldas-tools-2.4.2/etc/*
12. Edit the files in ldas-tools-2.4.2/etc/ to point to this particular ldas-tools installation
13. cd /opt/rtapps
14. rm ldas-tools
15. rm framecpp
16. ln –s ldas-tools-2.4.2 ldas-tools
17. ln –s ldas-tools-2.4.2 framecpp

Now do the 'ldconfig' steps as shown above

8 Large IFO System Upgrade Instructions

The following describes the steps required to upgrade a large IFO system from V2.9 to V3.0 Real-Time Code Generator (RCG) on the front-ends and DAQ. This involves multiple machines and operating systems.

8.1 Install RCG 3.0 software for Gentoo

Check out the tagged release from the repository and make it the default. We use an SVN checkout so we get version information.

1. Log in as 'controls' to the boot server (i.e. l1boot)
2. cd $RTCDSBASE/rtscore (this should take you to /opt/rtcds/rtscore - top-level checkout for advLigoRTS)

3. svn co https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-3.0
4. rm release (break link to old Gentoo RCG)
5. ln -s advLigoRTS-3.0 release (set link to new Gentoo RCG)
6. Logout out of your session, and then log back in. This will make the new release the default version.

8.2 Upgrade IFO front-ends to RCG 3.0

8.2.1 Set up new build area for front-ends

A new default front-end build area needs to be created and configured for the new RCG

1. Login to the boot server as 'controls'
2. cd $RTCDSROOT/rtbuild (this should take you to /opt/rtcds/<site>/<ifo>/rtbuild - top-level build area)
3. mkdir rt-3.0 (or similar)
4. cd rt-3.0
5. ${RCG_DIR}/configure - this will create Makefile, config.log, config.status files and doc,src folders
6. cd .. (this puts you back at $RTCDSROOT/rtbuild)
7. rm current (breaks link to old build area)
8. ln -s rt-3.0 current (set link to new build area)

8.2.2 Install updated drivers, scripts

8.2.2.1 Rebuild GDS libraries, awgtpman for front-ends

One should always rebuild awgtpman and the GDS libraries for RCG. Note this is done in the RCG checkout area.

1. log in as 'controls' to the boot server
2. cd $RTCDSROOT/target (the alias 'target' may take you here)
3. cd gds/bin
4. cp -p awgtpman bin_archive/awgtpman.rcg-2.9.x (to save the existing one)
5. rcgcode (should take you to /opt/rtcds/rtscore/release)
6. cd src/gds
7. make clean
8. make
9. cp awgtpman $RTCDSROOT/target/gds/bin/bin_archive/awgtpman.rcg-3.0 (copies in the new version)
10. target
11. cd gds/bin
12. cp -p bin_archive/awgtpman.rcg-3.0 awgtpman (to install the new one)

8.2.2.2 Update mbuf driver

Your existing mbuf driver is likely compatible, but to be safe, lets update it.

1. log in as 'controls' to the boot server
2. rcgcode (should take you to /opt/rtcds/rtscore/release)

3. cd src/drv/mbuf (to get to the mbuf directory)
4. make (to build new mbuf kernel object)
5. sudo make install
6. su root
7. cd /diskless/root//lib/modules/2.6.34.1/extra
8. cp -p mbuf.ko mbuf.ko.2.9 (to save the existing one)
9. cp /lib/modules/2.6.34.1/extra/mbuf.ko mbuf.ko.3.0 (copies in the new version)
10. cp -p mbuf.ko.3.0 mbuf.ko (makes the new version the active one)

8.2.2.3 Update IRIG-B driver

Your existing IRIG-B driver is likely compatible, but to be safe with updated leap second list, lets update it.

1. log in as 'controls' to the boot server
2. rcgcode (should take you to /opt/rtcds/rtscore/release)
3. cd src/drv/symmetricom (to get to the symmetricom directory)
4. make (to build new symmetricom kernel object)
5. sudo make install
6. su root
7. cd /diskless/root//lib/modules/2.6.34.1/extra
8. cp -p symmetricom.ko symmetricom.ko.2.9 (to save the existing one)
9. cp /lib/modules/2.6.34.1/extra/symmetricom.ko symmetricom.ko.3.0 (copies in the new version)
10. cp -p symmetricom.ko.3.0 symmetricom.ko (makes the new version the active one)

8.2.2.4 Update mx_stream driver

The mx_stream driver is updated in RCG 3.0 to support dual data streams. We need to build a new version so we will be ready to support it.

1. log in as 'controls' to the boot server
2. target
3. cd l1daqdc0
4. cp -p mx_stream bin_archive/mx_stream.2.9 (to save the existing one)
5. rcgcode
6. cd src/mx_stream
7. make clean
8. make
9. cp mx_stream $RTCDSROOT/target/l1daqdc0/bin_archive/mx_stream.rcg-3.0 (copies in the new version)
10. target
11. cd l1daqdc0
12. cp -p bin_archive/mx_stream.rcg-3.0 mx_stream (to install the new one)

8.2.2.5 Update mx_stream driver script

The new mx_stream code supports multiple instances of the driver, each going to a different data concentrator. Even on systems that have only one data concentrator, we have to add another command-line parameter to the call.

1. Login to boot server as 'root'
2. cd /diskless/root/etc/init.d
3. Edit the mx_stream file so the mx_stream command line has an added '–e 0' on it. For example:

```
start-stop-daemon --start --quiet -b -m --pidfile /var/log/mx_stream.pid --exec
/opt/rtcds/llo/l1/target/l1daqdc0/mx_stream -- -e 0 -r "$epnum" -W 0 -w 0 -s "$sys" -d l1daqdc0 –l
/opt/rtcds/llo/l2/target/l1daqdc0/mx_stream_logs/$hostname.log
```

8.2.2.6 Update scripts

There are some RCG-distributed scripts that need to be moved to the scripts area.

4. Login to boot server as 'controls'
5. rcgcode (alias to get to ${RCG_DIR})
6. cd src/epics/util
7. cp iniChk.pl ${RTCDSROOT}/scripts
8. cp fe_load_burt ${RTCDSROOT}/scripts
9. cp grdfiltdecode.py ${RTCDSROOT}/scripts

8.2.3 Build and install models

8.2.3.1 Clear out old IPC tables

The existing IPC table should be cleared so we can start afresh as we will be rebuilding all models. Replace 'L1' with the identifier of your IFO

1. Log into boot server as 'controls'
2. cd $RTCDSROOT/chans/ipc
3. mv L1.ipc L1_<date>.ipc
4. touch L1.ipc

8.2.3.2 Create new 'tmp' folder for filters

We need to add a 'tmp' directory to hold recent filter file updates for checks, edits

1. Log into boot server as 'controls'
2. cd $RTCDSROOT/chans
3. mkdir tmp

8.2.3.3 Rebuild all models

We will use the overall make command, but modifying our call the first time so that each build works once. This will fill the IPC list file.

1. Logout and back into the boot server as controls (to reset paths, aliases)
2. Use 'cdscode' to move to build area
3. make -i World (run make ignoring errors)
4. Check for errors in *_error.log files (grep ERROR *_error.log). Correct issues with ungrounded filter inputs, etc. in models
5. Inspect ipc file at ${RTCDSROOT}/chans/ipc/L1.ipc
6. cdscode
7. make World

8.2.3.4 Install all new models

1. cd to build area.
2. make installWorld

8.2.4 Restart all front-ends

Now we get to restart all the front-ends computers. This requires a full boot to get new kernels, start-up scripts to complete build, but without ignoring errors. This can be done manually or not

1. Log in to boot server
2. /etc/reboot_all_fes.sh

The text of this script is similar to 'shutdown_all_fes.sh'

```
echo "rebooting all front-ends"
/etc/allrt.sh 'sudo /sbin/init 6'
```

Wait patiently

8.2.5 Recover front-end models with non-running real-time model

If any of the front-end models only start partially (i.e. EPICS portion running, real-time is BAD), the likely cause is a bad/out-of-date safe.snap file. The best way to remedy this for each such model is to

1. Examine the GDS_TP screen to determine the DCUID/FEC number for that model
2. Set the BURT_RESTORE flag to 1 for that model

```
caput <IFO>:FEC-<DCUID>_BURT_RESTORE 1
```

3. Create a new safe.snap file, using either the SDF_RESTORE screen or the command-line utility makeSafeBackup <sub> <modelname>
4. Do as needed for all models on a front-end computer
5. Login to the front-end
6. sudo /etc/startWorld.sh (stops, then restarts all the models in the correct order)

8.3 Upgrade IFO DAQ to RCG 3.0

8.3.1 Update Gentoo DAQ machines

For RCG 3.0, we will be adding support for double-precision data. The core DAQ code already supports this data type, so this can be skipped if needed.

We have DAQ machines running Gentoo (2.6.35) as well as Ubuntu 12, so we will have to do this twice. First the Gentoo machines. Note that we had added steps to allow builds using arbitrary locations of EPICS and framecpp (see LIGO-T1500227). This is required to use the new ldas-tools builds above.

8.3.1.1 Stop DAQ processes on NDS server for Gentoo DAQ builds

We will do most of the Gentoo DAQ builds on one of the Gentoo DAQ machines. We usually choose an NDS server (i.e. l1daqnds0) as it is easiest to take offline. To free up memory for the build, we need to shut down the DAQ processes.

1. log in as 'controls' on an NDS server (i.e. l1daqnds0)

2. sudo /etc/init.d/monit stop (stop the monit process to keep from restarting daqd, nds)
3. sudo /etc/init.d/daqd_nds0 stop (stops daqd)
4. sudo /etc/init.d/nds_nds0 stop (stops nds)

8.3.1.2 Rebuild GDS libraries for Gentoo DAQ

We need to rebuild the GDS libraries to support the DAQ builds. This is due to changes there to support double-precision

1. log in as 'controls' on an NDS server (i.e. l1daqnds0)
2. rcgcode (should take you to /opt/rtcds/rtscore/release, which should be RCG 3.0 checout)
3. cd src/gds
4. make clean
5. make

8.3.1.3 Configure Gentoo daqd, nds build areas

We need to set up the GNU 'autoconf' tools to use info on the EPICS and framecpp packages to simplify builds. So we run '/.bootstrap' in the 'daqd' and 'nds' source directories

1. rcgcode (should take you to /opt/rtcds/rtscore/release, which should be RCG 3.0 checout)
2. cd src/daqd
3. ./bootstrap
4. cd ../nds
5. ./bootstrap
6. cd ../..

8.3.1.4 Set up new build area for Gentoo DAQ

For ease of support, we will use a dedicated build area for Gentoo DAQ software. This should work for all machines except the current frame-writers

1. log in as 'controls' on an NDS server (i.e. l1daqnds0)
2. cd $RTCDSROOT (this should take you to /opt/rtcds/<site>/<ifo>)
3. mkdir -p daqbuild
4. cd daqbuild
5. mkdir daq-3.0 (or similar)
6. cd daq-3.0
7. ${RCG_DIR}/configure - this will create Makefile, config.log, config.status files and doc,src folders
8. cd .. (this puts you back at $RTCDSROOT/daqbuild)
9. rm current (breaks link to old build area)
10. ln -s daq-3.0 current (set link to new build area)

8.3.1.5 Build the Gentoo data receiver executable (daqd) for trend-writer

We have a frame-writer specific build for daqd labeled 'fw' used for dedicated file writers. We currently use Gentoo machines as raw-trend file writers, and Ubuntu machines as frame file writers. Here we do the raw trend file writers.

1. log in as 'controls' on NDS server (i.e. l1daqnds0)
2. daqcode

3. make fw
4. cd build/fw
5. cp -p daqd ${RTCDSROOT}/target/l1daqtw0/bin_archive/daqd.rcg-3.0 (copies in the new version)
6. cp -p daqd ${RTCDSROOT}/target/l1daqtw1/bin_archive/daqd.rcg-3.0 (copies in the new version)

8.3.1.6 Build the Gentoo data receiver executable (daqd) for NDS servers

1. log in as 'controls' on NDS server
2. daqcode
3. make rcv
4. cd build/rcv
5. cp -p daqd ${RTCDSROOT}/target/l1daqnds0/bin_archive/daqd.rcg-3.0 (copies in the new version)
6. cp -p daqd ${RTCDSROOT}/target/l1daqnds1/bin_archive/daqd.rcg-3.0 (copies in the new version)

8.3.1.7 Build the Gentoo NDS executable (nds) for NDS

1. daqcode
2. make nds
3. cd build/nds
4. cp -p nds ${RTCDSROOT}/target/l1daqnds0/bin_archive/nds.rcg-3.0 (copies in the new version)
5. cp -p nds ${RTCDSROOT}/target/l1daqnds1/bin_archive/nds.rcg-3.0 (copies in the new version)

8.3.1.8 Build the Gentoo GDS broadcaster executable (daqd)

While still on the NDS server, we should build the GDS broadcaster executable.

1. daqcode
2. make bcst
3. cd build/bcst
4. cp -p daqd ${RTCDSROOT}/target/l1daqgds0/bin_archive/daqd.rcg-3.0 (copies in the new version)
5. cp -p daqd ${RTCDSROOT}/target/l1daqgds1/bin_archive/daqd.rcg-3.0 (copies in the new version)

8.3.1.9 Stop daqs on trend-writer, install new daqd executable, restart

1. log in as 'controls' to trend writer (i.e. l1daqfw0)
2. sudo /etc/init.d/monit stop
3. sudo /etc/init.d/daqd_tw0 stop
4. target
5. cd l1daqtw0
6. cp -p daqd bin_archive/daqd.rcg-2.9.x
7. cp -p bin_archive/daqd.rcg-3.0 daqd
8. sudo /etc/init.d/monit start

8.3.1.10 Install, run new daqd, nds executables on NDS server

1. log in as 'controls' to NDS server (i.e. l1daqnds0)
2. sudo /etc/init.d/monit stop
3. sudo /etc/init.d/daqd_nds0 stop
4. sudo /etc/init.d/nds_nds0 stop
5. target
6. cd l1daqnds0
7. cp -p daqd bin_archive/daqd.rcg-2.9.x
8. cp -p bin_archive/daqd.rcg-3.0 daqd
9. cp -p nds bin_archive/nds.rcg-2.9.x
10. cp -p bin_archive/nds.rcg-3.0 nds
11. sudo /etc/init.d/monit start

8.3.1.11 Install, run new daqd executables on GDS broadcaster

1. log in as 'controls' to GDS broadcaster (i.e. l1daqgds0)
2. sudo /etc/init.d/monit stop
3. sudo /etc/init.d/daqd_gds0 stop
4. target
5. cd l1daqgds0
6. cp -p daqd bin_archive/daqd.rcg-2.9.x
7. cp -p bin_archive/daqd.rcg-3.0 daqd
8. sudo /etc/init.d/monit start

8.3.1.12 Stop DAQ on data concentrator for IRIG-B driver update

We need to build daqd on data concentrator so it picks up the ‘mx’ driver files

1. log in as 'controls' to data concentrator (i.e. l1daqdc0)
2. sudo /etc/init.d/monit stop (stop the monit process to keep from restarting daqd, nds)
3. sudo /etc/init.d/daqd_dc0 stop (stops daqd)

8.3.1.13 Build Gentoo data concentrator executable

Now we build daqd for ‘dc’

1. log in as 'controls' to NDS server used for builds (i.e. l1daqnds0)
2. daqcode (Using an alias to get to /opt/rtcds/<site>/<ifo>/daqbuild/current)
3. make dc
4. cp -p build/dc/daqd ${RTCDSROOT}/target/l1daqdc0/bin_archive/daqd.rcg-3.0 (copies in the new version)
5. cp -p build/dc/daqd ${RTCDSROOT}/target/l1daqdc1/bin_archive/daqd.rcg-3.0 (copies in the new version)

8.3.1.14 Update IRIG-B driver on data concentrator

Your existing IRIG-B driver is likely compatible, but to be safe with updated leap second list, lets update it. This only needs to be done on the data concentrator

1. log in as 'controls' to data concentrator (i.e. l1daqdc0)
2. rcgcode (should take you to /opt/rtcds/rtscore/release)

3. cd src/drv/symmetricom (to get to the symmetricom directory)
4. make (to build new symmetricom kernel object)
5. sudo make install

8.3.1.15 Start new daqd on Gentoo data concentrator

1. log in as 'controls' to data concentrator (i.e. l1daqdc0)
2. target
3. cd l1daqdc0
4. cp -p daqd bin_archive/daqd.rcg-2.9.x (to preserve the existing one)
5. cp -p bin_archive/daqd.rcg-3.0 daqd (to install new one as active copy)
6. sudo /etc/init.d/monit start

8.3.2 Update Ubuntu DAQ machines

Now we need to do new builds for the Ubuntu 12 frame-writers. At present, these are working out of a special 'trunk' checkout, usually named 'ubutrunk'. Now we need to move to a release-oriented setup, using an 'uburelease' tag.

8.3.2.1 Install RCG 3.0 software for Ubuntu

Because much of the DAQ builds are done in the check-out directories, we want to have a separate area for the Ubuntu 12 framewriters (see LIGO-T1500458-v1). At this time, the Subversion releases on the Gentoo and Ubuntu machines are compatible, so we can do the check out on the boot server

1. Log in as 'controls' to the boot server (i.e. l1boot)
2. cd $RTCDSBASE/rtscore (this should take you to /opt/rtcds/rtscore - top-level checkout for advLigoRTS)
3. svn co https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-3.0 ubu-3.0
4. rm uburelease (break link to old Ubuntu RCG)
5. ln -s ubu-3.0 uburelease (set link to new RCG)

8.3.2.2 Update Ubuntu DAQ 'controls' user setup

Now that we are instituting an 'uburelease' directory for the checkout, we need to update the 'controls' account start-up on each such machine to point to the new location, updating the 'PRIVATE_RCG' definition. So on each such DAQ computer, ~controls/.bashrc should now look like:

```
#
#  point to ubuntu release RCG
#
export PRIVATE_RCG="/opt/rtcds/rtscore/uburelease"
#
# do the setup
source /opt/cdscfg/rtsetup.sh
```

8.3.2.3 Stop DAQ processes on secondary frame-writer for Gentoo DAQ builds

We will do the remaining Ubuntu DAQ builds on one of the framewriters. To free up memory for the build, we need to shut down the DAQ processes.

1. log in as 'controls' on secondary Ubuntu framewriter server (i.e. l1daqfw1)
2. sudo /etc/init.d/monit stop (stop the monit process to keep from restarting daqd)
3. sudo /etc/init.d/daqd stop (stops daqd)

8.3.2.4 Rebuild GDS libraries for Ubuntu DAQ

We need to rebuild the GDS libraries to support the DAQ builds. This is due to changes there to support double-precision

1. log in as 'controls' on secondary Ubuntu framewriter server (i.e. l1daqfw1)
2. rcgcode (should take you to /opt/rtcds/rtscore/uburelease, which should point to the RCG 3.0 checkout)
3. cd src/gds
4. make clean
5. make

8.3.2.5 Configure Ubuntu daqd, nds build areas

We need to set up the GNU 'autoconf' tools to use info on the EPICS and framecpp packages to simplify builds. So we run '/.bootstrap' in the 'daqd' and 'nds' source directories

1. rcgcode (should take you to /opt/rtcds/rtscore/uburelease)
2. cd src/daqd
3. ./bootstrap
4. cd ../nds
5. ./bootstrap
6. cd ../..

8.3.2.6 Set up new build area for Ubuntu DAQ

For ease of support, we will use a dedicated build area for Ubuntu DAQ software.

1. log in as 'controls' on secondary Ubuntu framewriter server (i.e. l1daqfw1)
2. cd $RTCDSROOT (this should take you to /opt/rtcds/<site>/<ifo>)
3. mkdir -p daqbuild
4. cd daqbuild
5. mkdir ubudaq-3.0 (or similar)
6. cd ubudaq-3.0
7. ${RCG_DIR}/configure - this will create Makefile, config.log, config.status files and doc,src folders
8. cd .. (this puts you back at $RTCDSROOT/daqbuild)
9. rm ubucurrent (breaks link to old build area)
10. ln -s ubudaq-3.0 ubucurrent (set link to new build area)

8.3.2.7 Build the Ubuntu data receiver executable (daqd) for frame-writer

We have a frame-writer specific build for daqd labeled 'fw' used for dedicated file writers. We currently use Gentoo machines as raw-trend file writers, and Ubuntu machines as frame file writers. Here we do the frame file writers

1. log in as 'controls' on secondary Ubuntu framewriter (i.e. l1daqfw1)
2. cd $RTCDSROOT

3. cd daqbuild/ubucurrent
4. make fw
5. cd build/fw
6. cp -p daqd ${RTCDSROOT}/target/l1daqfw0/bin_archive/daqd.rcg-3.0 (copies in the new version)
7. cp -p daqd ${RTCDSROOT}/target/l1daqfw1/bin_archive/daqd.rcg-3.0 (copies in the new version)

8.3.2.8 Install new daqd executable, restart on secondary frame-writer

Here the frame-writing was already disabled.

1. Log in to secondary Ubuntu framewriter i.e. l1daqfw1) as controls
2. sudo ldconfig (to update LD paths)
3. target
4. cd l1daqfw1
5. cp -p daqd bin_archive/daqd.rcg-2.9.x
6. cp -p bin_archive/daqd.rcg-3.0 daqd
7. sudo /etc/init.d/monit start

8.3.2.9 Stop frame-writing on primary frame-writer, install new daqd executable, restart

1. Log in to primary framewriter (i.e. l1daqfw0) as controls
2. sudo /etc/init.d/monit stop
3. sudo /etc/init.d/daqd stop
4. sudo ldconfig (to update LD paths)
5. target
6. cd l1daqfw0
7. cp -p daqd bin_archive/daqd.rcg-2.9.x
8. cp -p bin_archive/daqd.rcg-3.0 daqd
9. sudo /etc/init.d/monit start

9 Dedicated Framebuilder System Upgrade Instructions

The following describes the steps required to upgrade smaller systems with multiple front-ends but a single frame-builder computer from V2.9 to V3.0.

9.1 Install RCG 3.0 software for front-ends

Check out the tagged release from the repository and make it the default. We use an SVN checkout so we get version information.

1. Log in as 'controls' to the boot server (i.e. m1boot)
2. cd $RTCDSBASE/rtscore (this should take you to /opt/rtcds/rtscore - top-level checkout for advLigoRTS)
3. svn co https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-3.0
4. rm release (break link to old Gentoo RCG)
5. ln -s advLigoRTS-3.0 release (set link to new Gentoo RCG)

6. Logout out of your session, and then log back in. This will make the new release the default version.

9.2 Upgrade front-ends to RCG 3.0

On these systems, the front-ends are configured as they are on the large IFO systems, using the MX protocol to send data from the multiple front-ends to the single DAQ computer (framebuilder). So simply follow the instructions in Section 8.2 above.

9.3 Upgrade framebuilder to RCG 3.0

For RCG 3.0, we will be adding support for double-precision data. The core DAQ code already supports this data type, so this can be skipped if needed. Note that we had added steps to allow builds using arbitrary locations of EPICS and framecpp (see LIGO-T1500227). This is required to use the new ldas-tools builds above.

Here we assume the framebuilder is running a different OS or at least a different version, so it needs a separate checkout and build area.

9.3.1.1 Install RCG 3.0 software for framebuilder

Because much of the DAQ builds are done in the check-out directories, we want to have a separate area for the framebuilder. (see LIGO-T1500458-v1).

1. Log in as 'controls' to the framebuilder (i.e. m1fb0)
2. cd $RTCDSBASE/rtscore (this should take you to /opt/rtcds/rtscore - top-level checkout for advLigoRTS)
3. svn co https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-3.0 fb-3.0
4. rm fbrelease (break link to old Ubuntu RCG)
5. ln -s fb-3.0 fbrelease (set link to new RCG)

9.3.1.2 Update DAQ 'controls' user setup

Now that we are instituting an 'fbrelease' directory for the checkout, we need to update the 'controls' account start-up on the framebuilder to point to the new location, updating the 'PRIVATE_RCG' definition. So on the framebuilder, ~controls/.bashrc should now look like:

```
#
#  point to framebuilder release RCG
#
export PRIVATE_RCG="/opt/rtcds/rtscore/fbrelease"
#
# do the setup
source /opt/cdscfg/rtsetup.sh
```

9.3.1.3 Stop DAQ processes on framebuilder for DAQ builds

We will do the DAQ builds on the framebuilder. To free up memory for the build, we need to shut down the DAQ processes.

1. log in as 'controls' on framebuilder (i.e. m1fb0)
2. sudo /etc/init.d/monit stop (stop the monit process to keep from restarting daqd)
3. sudo /etc/init.d/daqd stop (stops daqd)

9.3.1.4 Rebuild GDS libraries for DAQ

We need to rebuild the GDS libraries to support the DAQ builds. This is due to changes there to support double-precision

1. rcgcode (should take you to /opt/rtcds/rtscore/fbrelease, which should point to the RCG 3.0 checkout)
2. cd src/gds
3. make clean
4. make

9.3.1.5 Configure daqd, nds build areas

We need to set up the GNU 'autoconf' tools to use info on the EPICS and framecpp packages to simplify builds. So we run '/.bootstrap' in the 'daqd' and 'nds' source directories

1. rcgcode (should take you to /opt/rtcds/rtscore/fbrelease)
2. cd src/daqd
3. ./bootstrap
4. cd ../nds
5. ./bootstrap
6. cd ../..

9.3.1.6 Set up new build area for DAQ

For ease of support, we will use a dedicated build area for DAQ software.

1. log in as 'controls' on framebuilder (m1fb0)
2. cd $RTCDSROOT (this should take you to /opt/rtcds/<site>/<ifo>)
3. mkdir -p daqbuild
4. cd daqbuild
5. mkdir daq-3.0 (or similar)
6. cd daq-3.0
7. ${RCG_DIR}/configure - this will create Makefile, config.log, config.status files and doc,src folders
8. cd .. (this puts you back at $RTCDSROOT/daqbuild)
9. rm current (breaks link to old build area)
10. ln -s daq-3.0 current (set link to new build area)

9.3.1.7 Build framebuilder 'daqd', 'nds' executables

Now we build daqd, nds executables. For a single-machine frame-builder, we use 'mx-symm'

1. log in as 'controls' to framebuilder (m1fb0)
2. daqcode (Using an alias to get to /opt/rtcds/<site>/<ifo>/daqbuild/current)
3. make mx-symm
4. cp -p build/mx-symm/daqd ${RTCDSROOT}/target/m1fb0/bin_archive/daqd.rcg-3.0 (copies in the new version)
5. make nds
6. cp -p build/nds/nds ${RTCDSROOT}/target/m1fb0/bin_archive/nds.rcg-3.0 (copies in the new version)

9.3.1.8 Update IRIG-B driver on framebuilder

Your existing IRIG-B driver is likely compatible, but to be safe with updated leap second list, lets update it. This only needs to be done on the data concentrator

1. log in as 'controls' to framebuilder (m1fb0)
2. rcgcode (should take you to /opt/rtcds/rtscore/fbrelease)
3. cd src/drv/symmetricom (to get to the symmetricom directory)
4. make (to build new symmetricom kernel object)
5. sudo make install

9.3.1.9 Start new daqd, nds on framebuilder

1. log in as 'controls' to framebuilder (m1fb0)
2. target
3. cd m1fb0
4. cp -p daqd bin_archive/daqd.rcg-2.9.x (to preserve the existing one)
5. cp -p bin_archive/daqd.rcg-3.0 daqd (to install new one as active copy)
6. cp -p nds bin_archive/nds.rcg-2.9.x (to preserve the existing one)
7. cp -p bin_archive/nds.rcg-3.0 nds (to install new one as active copy)
8. sudo /etc/init.d/monit start

10 Standalone System Upgrade Instructions

The following describes the steps required to upgrade standalone system from V2.9 to V3.0 Real-Time Code Generator (RCG). Here, the same machine does real-time models and the DAQ.

10.1 Install RCG 3.0 software

Check out the tagged release from the repository and make it the default. We use an SVN checkout so we get version information.

1. Log in as 'controls' to the stand-alone machine
2. cd $RTCDSBASE/rtscore (this should take you to /opt/rtcds/rtscore - top-level checkout for advLigoRTS)
3. svn co https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-3.0
4. rm release (break link to old Gentoo RCG)
5. ln -s advLigoRTS-3.0 release (set link to new Gentoo RCG)
6. Logout out of your session, and then log back in. This will make the new release the default version.

10.2 Set up new build area

A new default build area needs to be created and configured for the new RCG

1. cd $RTCDSROOT/rtbuild (this should take you to /opt/rtcds/<site>/<ifo>/rtbuild - top-level build area)
2. mkdir rt-3.0 (or similar)
3. cd rt-3.0
4. ${RCG_DIR}/configure - this will create Makefile, config.log, config.status files and doc,src folders

5. cd .. (this puts you back at $RTCDSROOT/rtbuild)
6. rm current (breaks link to old build area)
7. ln -s rt-3.0 current (set link to new build area)

10.3 Upgrade real-time models to RCG 3.0

10.3.1 Install updated drivers, scripts

10.3.1.1 Rebuild GDS libraries, awgtpman

One should always rebuild awgtpman and the GDS libraries for RCG. Note this is done in the RCG checkout area.

1. cd $RTCDSROOT/target (the alias 'target' may take you here)
2. cd gds/bin
3. cp -p awgtpman bin_archive/awgtpman.rcg-2.9.x (to save the existing one)
4. rcgcode (should take you to /opt/rtcds/rtscore/release)
5. cd src/gds
6. make clean
7. make
8. cp awgtpman $RTCDSROOT/target/gds/bin/bin_archive/awgtpman.rcg-3.0 (copies in the new version)
9. target
10. cd gds/bin
11. cp -p bin_archive/awgtpman.rcg-3.0 awgtpman (to install the new one)

10.3.1.2 Update mbuf driver

Your existing mbuf driver is likely compatible, but to be safe, let's update it.

1. rcgcode (should take you to /opt/rtcds/rtscore/release)
2. cd src/drv/mbuf (to get to the mbuf directory)
3. make (to build new mbuf kernel object)
4. sudo make install

10.3.1.3 Update IRIG-B driver

Your existing IRIG-B driver is likely compatible, but to be safe with updated leap second list, lets update it.

1. rcgcode (should take you to /opt/rtcds/rtscore/release)
2. cd src/drv/symmetricom (to get to the symmetricom directory)
3. make (to build new symmetricom kernel object)
4. sudo make install

10.3.1.4 Update scripts

There are some RCG-distributed scripts that need to be moved to the scripts area.

1. Login to boot server as 'controls'
2. rcgcode (alias to get to ${RCG_DIR})
3. cd src/epics/util

4. cp iniChk.pl ${RTCDSROOT}/scripts
5. cp fe_load_burt ${RTCDSROOT}/scripts
6. cp grdfiltdecode.py ${RTCDSROOT}/scripts

10.3.2 Build and install models

10.3.2.1 Clear out old IPC tables

The existing IPC table should be cleared so we can start afresh as we will be rebuilding all models. Replace 'S1' with the identifier of your IFO

1. cd $RTCDSROOT/chans/ipc
2. mv S1.ipc S1_<date>.ipc
3. touch S1.ipc

10.3.2.2 Create new 'tmp' folder for filters

We need to add a 'tmp' directory to hold recent filter file updates for checks, edits

1. cd $RTCDSROOT/chans
2. mkdir tmp

10.3.2.3 Rebuild all models

We will use the overall make command, but modifying our call the first time so that each build works once. This will fill the IPC list file.

1. Use 'cdscode' to move to build area
2. make -i World (run make ignoring errors)
3. Check for errors in *_error.log files (grep ERROR *_error.log). Correct issues with ungrounded filter inputs, etc. in models
4. Inspect ipc file at ${RTCDSROOT}/chans/ipc/S1.ipc
5. cdscode
6. make World

10.3.2.4 Install new models

1. cd to build area.
2. make installWorld

10.3.3 Restart front-end models

This can be done manually or not

/etc/startWorld.sh

Wait patiently

10.3.4 Recover front-end models with non-running real-time model

If any of the front-end models only start partially (i.e. EPICS portion running, real-time is BAD), the likely cause is a bad/out-of-date safe.snap file. The best way to remedy this for each such model is to

1. Examine the GDS_TP screen to determine the DCUID/FEC number for that model
2. Set the BURT_RESTORE flag to 1 for that model

```
caput <IFO>:FEC-<DCUID>_BURT_RESTORE 1
```

3. Create a new safe.snap file, using either the SDF_RESTORE screen or the command-line utility makeSafeBackup <sub> <modelname>
4. Do as needed for all models on a front-end computer
5. Login to the front-end
6. sudo /etc/startWorld.sh (stops, then restarts all the models in the correct order)

10.4 Update DAQ software to RCG 3.0

10.4.1 Stop DAQ processes

To free up memory for the build, we need to shut down the DAQ processes. The following assumes that 'monit' is being used to start/stop the DAQ processes. On older machines that use 'inittab', a more severe procedure is needed.

1. log in as 'controls' to the machine
2. sudo /etc/init.d/monit stop (stop the monit process to keep from restarting daqd, nds)
3. sudo /etc/init.d/daqd stop (stops daqd)
4. sudo /etc/init.d/nds stop (stops nds)

10.4.2 Configure daqd, nds build areas

We need to set up the GNU 'autoconf' tools to use info on the EPICS and framecpp packages to simplify builds. So we run '/.bootstrap' in the 'daqd' and 'nds' source directories

1. rcgcode (should take you to /opt/rtcds/rtscore/release, which should be RCG 3.0 checout)
2. cd src/daqd
3. ./bootstrap
4. cd ../nds
5. ./bootstrap
6. cd ../..

10.4.3 Reconfigure build area

We need to update the configuration of the build area after the daqd, nds bootstraps

1. Use 'cdscode' to move to build area
2. ${RCG_DIR}/configure - this will create Makefile, config.log, config.status files and doc,src folders

10.4.4 Build DAQ executable (daqd)

We build the 'standiop' version of daqd. Here we assume the DAQ executables run out of the 'fb' target area.

1. daqcode (Using an alias to get to /opt/rtcds/<site>/<ifo>/daqbuild/current)
2. make standiop
3. cp -p build/standiop/daqd ${RTCDSROOT}/target/fb/bin_archive/daqd.rcg-3.0 (copies in the new version)

10.4.5 Build NDS executable (nds)

Same assumption about ‘’fb’ target area.

1. daqcode
2. make nds
3. cd build/nds
4. cp -p nds ${RTCDSROOT}/target/fb/bin_archive/nds.rcg-3.0 (copies in the new version)

10.4.6 Install, run new daqd,nds executables

1. target
2. cd fb
3. cp -p daqd bin_archive/daqd.rcg-2.9.x
4. cp -p bin_archive/daqd.rcg-3.0 daqd
5. cp -p nds bin_archive/nds.rcg-2.9.x
6. cp -p bin_archive/nds.rcg-3.0 nds
7. sudo /etc/init.d/monit start

11 Build new dataviewer on workstations

We need a new version of dataviewer that supports the double data type. The source code is distributed as part of the RCG. We need to check out RCG 3.0 on a workstation, then build and install it. We then change a soft-link to make it the default. Note that we need to rebuild stuff in the 'gds' folder first

1. Log into a workstation
2. Navigate to a build directory. At LLO, I use /ligo/cds/projects/advLigoRTS
3. svn co https://redoubt.ligo-wa.caltech.edu/svn/advLigoRTS/tags/advLigoRTS-3.0
4. cd advLigoRTS-3.0
5. cd src/gds
6. make clean
7. make
8. cd ../dv
9. make clean
10. make
11. su controls (or whatever account has privileges to install in $APPSROOT)
12. make install
13. cd $APPSROOT
14. Check that a new dv-3.0 directory has been created
15. rm dv (remove old link)
16. ln -s dv-3.0 dv (set link to new RCG)