LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T1400570-v7 *LIGO* June 22, 2015

Real-time Code Generator (RCG)
Version 2.9.3 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 V2.8 to V2.9.
- Provide installation instructions for the new V2.9 release.

2 Release History

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

- branch-2.9: Initial release for testing only. (October, 2014)
- advLigoRTS-2.9 (January, 2015) – Release notes v4 and earlier apply
- advLigoRTS-2.9.1 (March, 2015) – Initial documentation version 5 of this document.
- advLigoRTS-2.9.2 (June, 2015) – RFM IPC sender timing option

3 New Features

3.1 Data Acquisition (DAQ)

With this version, support for UINT32 data types, both for fast and EPICS channels, is incorporated. This includes a bug fix to the main data acquisition code (daqd) that now properly tags the data as UINT32. A new part has also been added to the RCG MATLAB library to support EPICS UINT32 data types.

3.1.1 Support for EPICS UINT32 Data Types

As shown below, a new part has been added to support acquisition and archival of EPICS channels as UINT32 type. This would be used in a control model the same as the standard EPICS output part, but now produces a UINT32 data type at the output and to the DAQ system. As with other EPICS channels, these will be automatically recorded by the DAQ system at 16Hz.

Figure 1: EPICS UINT32 PART

3.1.2 Support for assigning engineering units (EGU).

The assignment of engineering units (EGU) to DAQ channels is also supported in this release.

3.1.2.1 Fast Channel EGU Assignment

Assignment of EGU for fast data channels is done by adding an alphanumeric string within the DAQ channel part list. In the following example, the EGU of "Volts" is assigned to the channel DAQ_FILTER_4_OUT. When parsing this list, the RCG takes any alphanumeric string after the channel name to be the EGU for that channel. The exceptions are uint32 and int32, which the RCG interprets to be the data type.

```
#DAQ Channels

DAC_FILTER_1_OUT
DAC_FILTER_2_OUT
DAC_FILTER_3_OUT
DAC_FILTER_4_OUT* 512 Volts
ADC_FILTER_5_OUT uint32 512
DAC_FILTER_5_OUT uint32 256
```

3.1.2.2 EPICS Channel EGU Assignment

The assignment of EGU to EPICS channels is done by adding the EGU field assignment to the Description property of the EPICS part. To do this, select the part and then its Block Properties. The window appears, as in the following example. To the Description, add field(EGU,"units"), where units = desired EGU.

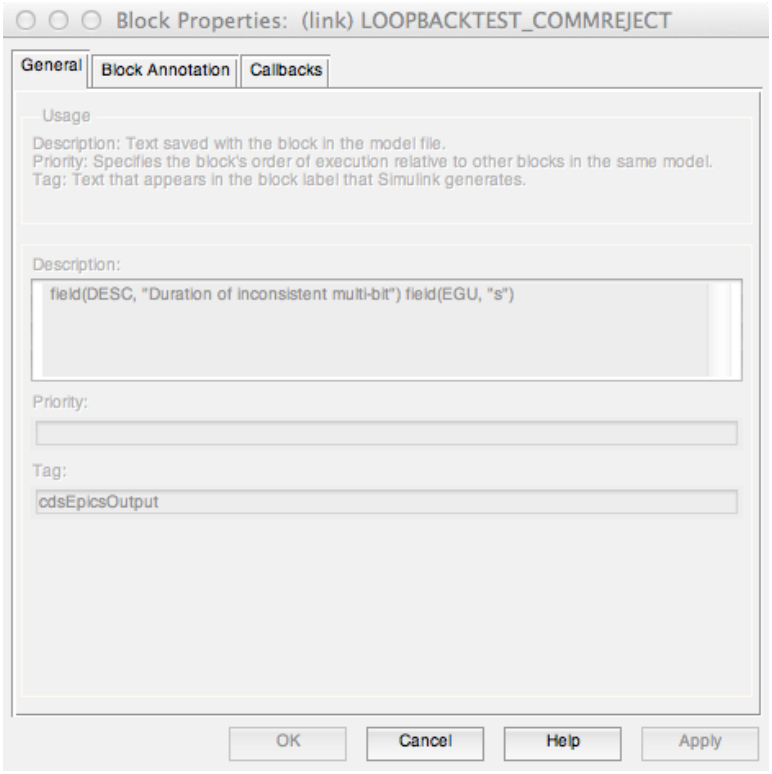

Figure 2: Example EGU assignment in Block Properties

3.2 Monitoring of Control Settings

The major change in V2.9 is the ability for the EPICS code, running on the Front End Computer (FEC) to directly read in EPICS Back Up and Restore Tool (BURT) files and monitor the settings for changes. This change was made primarily for two reasons:

1. Performance. For large control models, such as for ASC, BURT files could take in access of 20 seconds to load using EPICS Channel Access (CA). By having the EPICS code, generated by the RCG, directly read the file and using EPICS database access routines, the time to read and load settings was reduced by as much as a factor of 100.
2. LIGO CDS contains on the order of 100,000 control settings, of which about 80% are set once and not normally changed. However, if a setting is changed unexpectedly, for any number of possible reasons, it would be very difficult, at best, to track it down. Therefore, with the FEC now reading in the BURT file, it can also now monitor settings and report if and when a setting has changed.

This new software is commonly referred to as the SDF (Setpoint Definition File) code, after the new file format developed to apply reference settings. Complete details of SDF can now be found in a separate document, RCG SDF Software LIGO-T1500115.

3.3 New Filter Module Switch Setting Readout EPICS Channel

Filter module switch settings are done via two EPICS records:

- FilterModuleName_**SW1S**
- FilterModuleName_**SW2S**

Interpretation of these settings is difficult, at best, for operators ie matching numbers to individual switch settings. Therefore, in V2.9.1, a new, human readable, EPICS string variable has been added for every filter module, namely:

- FilterModuleName_**SWSTR**

For every filter module switch that is ON, this string contains a one or two character representation, with a comma (,) in between, as defined:

- IN = Input Switch
- OF = Offset Switch
- 1 thru 10 = Filter switch 1 thru 10
- LT = Limit Switch
- OT = Output Switch
- DC = Decimation Switch
- HD = Hold Output Switch

Some examples:

- Input, Output and Dec Switches ON, all others OFF, SWSTR = IN,OT,DC
- Input, F1, F2, Output and Dec Switches ON, all others OFF, SWSTR = IN,1,2,OT,DC

3.4 ADC and DAC Overflow Monitoring

In past releases, the monitoring of ADC/DAC channel overflows was via a single EPICS channel for each ADC channel, *IFO*:FEC-*DCUID*_ADC_OVERFLOW_*CARDNUM*_*CHANNUM*, and

similarly a single EPICS channel for each DAC channel. There was a compile option where the reporting could be set to either:

- Number of overflows detected per second, reset each second.
- Total number of overflows, reset on demand by the OVERFLOW_RESET.

In this version of code, both the overflows/second and total overflows are provided together, with total overflows (*IFO*:FEC-*DCUID*_ADC_OVERFLOW_**ACC**_*CARDNUM_CHANNUM*) updated continuously and available in the DAQ system at 16Hz. The auto generated ADC/DAC monitor screens have also been updated to reflect these new channels, as shown below.

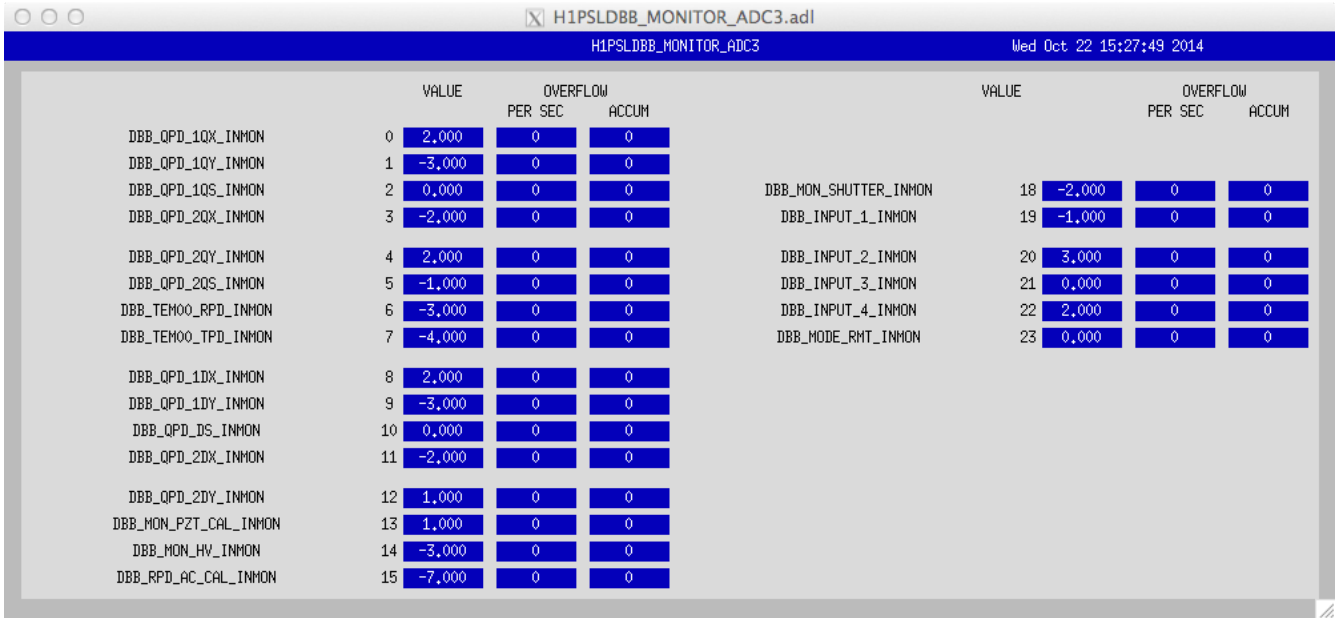

Figure 3 New ADC Monitor screen with Overflows

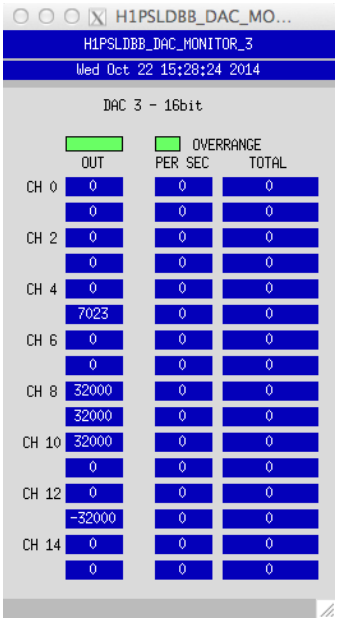

Figure 4 New DAC Monitor Screen with Overflows

3.5 Inter-Process Communications (IPC)

A special release, RCG V2.9.2, was made to address specific IPC errors between end and corner station computers. The issue was that a specific control model could not consistently complete its code cycle in time to reach the receiver in time for its next code cycle, due to the 20usec transmission delay of the 4km fiber, thereby resulting in IPC errors being reported. In the case being addressed, it was also not required, for control purposes, that the IPC being sent arrive in time for the receiver next code cycle, but would be sufficient for it to arrive one receiver code cycle later, or one cycle delayed.

To accommodate this, the RFM IPC sender code was modified with an option to properly timestamp and queue a send IPC such that it would arrive and be properly tagged for use at the receiver one cycle later than usual This option is invoked in the sender control model as an added line to the Parameter block:

rfm_delay = 1

Some implementation notes:

- This only works with RFM type IPC ie not PCIe network or shared memory IPC.
- When used, this definition applies to all RFM IPC sender parts in the model ie cannot be invoked on a part by part basis.

4 Bug Fixes

4.1 RCG V2.9

Bugzilla Number	Description	Comments
505	Add warning bit to StateWord if coeff or DAQ configuration files have changed and not yet loaded.	
553	Bit2word part not correctly wiring channel 15	Fixed in RCG 2.8.4
556	Fix WFSPHASE part to allow DAQ to record user settings.	
563	Update IRIG-B driver to handle dates in 2014	Used at MIT
595	No compile errors for FilterControl2 parts with unconnected inputs	Error now reported during RCG compilation
596	cdsRampMuxMatrix: channels not-ideally named	Fixed in RCG 2.8.3
597	cdsRampMuxMatrix: MEDM screens unwieldy	Fixed in RCG 2.8.3
621	Runtime errors when ADC parts used incorrectly in model	Error now reported during RCG compilation
658	Fix oscillator ramping behavior when changing frequencies	Fixed in RCG 2.8.3
660	Add daqd thread info to log	For debugging
670	DACKILL reset button caches the reset request	Fixed in RCG 2.8.4
662	Fix to PHASE part to allow DAQ to record user settings	
663	Load matrix button is intermittent	Fixed in RCG 2.8.4
683	Mean trend data for integer data displays as 0	Fixed in RCG 2.8.6
689	Correct leap-seconds in spectracomGPS.c	Corrected from MIT visit
690	Added framecpp location to NDS builds	
701	Update front-end boot scripts	Now same start/kill scripts created by RCG 2.9
703	Compilation Errors if signal sources directly to subsystem outputs	Fixed in RCG 2.8.6
722	Filter Mux Matrix MEDM in wrong directory	Fixed in RCG 2.8.6
729	Fix to TRUE RMS part if input = 0.0	
732	Set AUTOCAL of 18 bit DAC modules as default	Fixed in RCG 2.8.7

751	Install script modified for start/kill scripts	
756	Added monitoring of filter module DEC/HOLD switch settings for DAQ	
758	Load Ramp Mux Matrix issue	Fixed in RCG 2.8.7
762	DAQ error bit value increased to max = 4MB/sec	
772,773	SDF file save issues with initial branch-2.9 code	
776	Intermittent operation of EXC parts	Fixed in RCG 2.8.7

4.2 RCG V2.9.1

Bugzilla Number	Description	Comments
743/791	Increase ADC0 allowable wait	
751	Modified install script.	
759	Fix DAQ calculation of science frame 'Fast Chans' on stand-alone	
763	rampMuxMatrix fix to screen size and background color for fractional settings	
790	SDF Modifications	See T1500115
792	Wait for RT startup to complete in start scripts	
796	Add paging to SDF TABLE display	See T1500115
799	Added a third output, STATE, to the DacKillIop part.	
802	Changed SDF reporting records from string type to waveform type to allow >40 character strings to be reported to SDF MEDM screens.	
803	Modified Makefile.linux to not add SDF EPICS variables to the autoBurt.req file.	
810	Code not properly resetting SW1S and SW2S for FMC parts when in local mode. Operation of the FMC parts was not affected by this bug, but caused issues with SDF reporting.	
811	NDS1 Version 12.2: Fix byte-ordering in 'status channels 2'	

4.3 RCG V2.9.2

Bugzilla Number	Description	Comments
	RFM IPC Sender option	As described in section 3.5 of this document.

4.4 RCG V2.9.3

Bugzilla Number	Description	Comments
850	Modifications to SDF Table	
860	Modification to EPICS sequencer to pass filter module ramp times to real-time code before offset/gain values.	

5 Installation Instructions

The following describes the steps required to upgrade a system from V2.8 to V2.9 Real-Time Code Generator (RCG) on the front-ends and DAQ.

5.1 Get RCG 2.9 release

5.1.1 Install RCG 2.9 software

Check out the tagged release from the repository and make it the default. (We use an 'export' from Subversion so we only get the files and not hooks to check in updates)

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/branches/branch-2.9
4. rm release (break link to old RCG)
5. ln -s branch-2.9 release (set link to new RCG)
6. Logout out of your session, and then log back in. This will make the new release the default version.

5.2 Update DAQ machines

For RCG 2.9, we will be adding support to the DAQ for unsigned 32-bit integers (UINT32). It is better to start the upgrade process with the DAQ and clients, and then work towards the front-ends. This prevents changes to UINT32 handling on the front-ends from breaking the DAQ.

5.2.1 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 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-2.9 (or similar)
6. cd daq-2.9
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-2.9 current (set link to new build area)

5.2.2 Stop DAQ processes on NDS server for DAQ builds

We will do the remaining DAQ builds on one of the 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)

5.2.3 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 UINT32

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 2.9 checout)
3. cd src/gds
4. make clean
5. make

5.2.4 Build and install data concentrator executable (daqd)

We need to stop existing one to speed the build

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)
4. daqcode (Using an alias to get to /opt/rtcds/<site>/<ifo>/daqbuild/current)
5. make dc
6. cp -p build/dc/daqd ${RTCDSROOT}/target/l1daqdc0/bin_archive/daqd.rcg-2.9 (copies in the new version)
7. target
8. cd l1daqdc0
9. cp -p daqd bin_archive/daqd.rcg-2.8.2 (to preserve the existing one)
10. cp -p bin_archive/daqd.rcg-2.9 daqd (to install new one as active copy)
11. sudo /etc/init.d/monit start

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

We have a frame-writer specific build for daqd labeled 'fw'

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/l1daqfw0/bin_archive/daqd.rcg-2.9 (copies in the new version)
6. cp -p daqd ${RTCDSROOT}/target/l1daqfw1/bin_archive/daqd.rcg-2.9 (copies in the new version)

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

1. Log in to framewriter (i.e. l1daqfw0) as controls
2. sudo /etc/init.d/monit stop
3. sudo /etc/init.d/daqd_fw0 stop
4. target

5. cd l1daqfw0
6. cp -p daqd bin_archive/daqd.rcg-2.8.2
7. cp -p bin_archive/daqd.rcg-2.9 daqd
8. sudo /etc/init.d/monit start

5.2.7 Build the 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-2.9 (copies in the new version)
6. cp -p daqd ${RTCDSROOT}/target/l1daqnds1/bin_archive/daqd.rcg-2.9 (copies in the new version)

5.2.8 Build the 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-2.9 (copies in the new version)
5. cp -p nds ${RTCDSROOT}/target/l1daqnds1/bin_archive/nds.rcg-2.9 (copies in the new version)

5.2.9 Build the GDS broadcaster executable (daqd)

While still on the NDS server

1. daqcode
2. make bcst
3. cd build/bcst
4. cp -p daqd ${RTCDSROOT}/target/l1daqgds0/bin_archive/daqd.rcg-2.9 (copies in the new version)

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

1. Log in to NDS server (i.e. l1daqnds0) as controls
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-2.8.2
8. cp -p bin_archive/daqd.rcg-2.9 daqd
9. cp -p nds bin_archive/nds.rcg-2.8.2
10. cp -p bin_archive/nds.rcg-2.9 nds
11. sudo /etc/init.d/monit start

5.2.11 Install, run new daqd executables on GDS broadcaster

1. Log in to GDS broadcaster (i.e. l1daqgds0) as controls
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-2.8.2
7. cp -p bin_archive/daqd.rcg-2.9 daqd
8. sudo /etc/init.d/monit start

5.3 Build new dataviewer on workstations

We need a new version of dataviewer that supports the UINT32 data type. The source code is distributed as part of the RCG. We need to check out RCG 2.9 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/branches/branch-2.9
4. cd branch-2.9
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-2.9.1 directory has been created
15. rm dv (remove old link)
16. ln -s dv-2.9 dv (set link to new RCG)

5.4 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-2.9 (or similar)
4. cd rt-2.9
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-2.9 current (set link to new build area)

5.5 Install updated drivers, scripts

5.5.1 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-2.8.2 (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-2.9 (copies in the new version)
10. target
11. cd gds/bin
12. cp -p bin_archive/awgtpman.rcg-2.9 awgtpman (to install the new one)

5.5.2 Update iniChk.pl script

An updated iniChk.pl script needs to be moved to the scripts area. This script is used by the EPICS sequencer to check the DAQ configuration files for correctness prior to passing information to the real-time code.

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

5.5.3 Install new scripts (fe_load_burt, grdfiltdecode.py)

To support the new SDF file MEDM interface, two scripts (fe_load_burt, grdfiltdecode.py) 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 fe_load_burt ${RTCDSROOT}/scripts
5. cp grdfiltdecode.py ${RTCDSROOT}/scripts

5.6 Update front-end boot scripts

For RCG 2.9, the per-model start/stop scripts created in the RCG build process no longer match the same sections in the original front-end boot sequence. We need to install new boot scripts for the front-ends to use. The new scripts directly use the RCG-generated scripts, so they will no longer diverge.

This change is compatible with earlier RCG releases. It removes awgtpman start/stop from monit, so it is only done in the start/stop scripts, eliminating the potential for duplicate 'awgtpman' processes. It also moves front-end model EPICS startup until after the dolphin startup checks. This should actually make full-system-restarts smoother.

5.6.1 Copy in updated boot scripts

1. Log into boot server as 'controls'
2. Use 'rcgcode' to move to RCG 2.9 checkout area at '/opt/rtcds/rtscore/release'
3. cd src/feboot
4. sudo cp start_models.sh /diskless/root/etc
5. sudo cp kill_models.sh /diskless/root/etc
6. sudo cp monitrc /diskless/root/etc
7. sudo cp run_stdenv.sh /diskless/root/etc
8. sudo cp startWorld.sh /diskless/root/etc

5.6.2 Modify existing 'rc.local' script

This is the script that controls how the front-end boots. An updated one is provided in src/feboot. However, it is set up to use the new Dolphin multiple-netmanager scheme (See LIGO-T1300518). This is recommended for all sites with multiple Dolphin switches. However, if your site is not configured that way, here are the changes to 'rc.local.'

The existing code does front-end EPICS, Dolphin, front-end real-time, then awgtpman monit:

```
# Start all configured control systems
/etc/start_epics.sh

# Start OpenMX stream to DAQ
/etc/init.d/mx_stream start

# Wait for Dolphin to initialize on all nodes (if present)
if /etc/dolphin_config.sh
then
    /etc/dolphin_wait
fi

# Run front-ends
/etc/start_fes.sh

# Run IOP awgtpman
/etc/init.d/awgtpman_iop start

# Configure monit to use write-able area (/var/log)
mkdir -p /var/run/monit.d
touch /var/run/monit.d/empty
ln -snf /opt/monit/monit.$HOSTNAME.id /var/log/.monit.id

# Generate awgtpman init.d scripts and monit rules for all configured slaves
/etc/gen_awgtpman_scripts.sh

# Run service monitoring
/etc/init.d/monit start
```

We need to change this to use the new scripts, and remove awgtpman monit stuff

```
# Wait for Dolphin to initialize on all nodes (if present)
if /etc/dolphin_config.sh
then
    /etc/dolphin_wait
fi

# Start OpenMX stream to DAQ
/etc/init.d/mx_stream start

# start front-end models
/etc/start_models.sh

# Configure monit to use write-able area (/var/log)
mkdir -p /var/run/monit.d
touch /var/run/monit.d/empty
ln -snf /opt/monit/monit.$HOSTNAME.id /var/log/.monit.id

# Run service monitoring
/etc/init.d/monit start
```

5.6.3 Test new boot scripts

Now we want to test that this works.

1. Log onto a non-critical front-end (such as a SUSAUX machine)
2. sudo shutdown –r now (to reboot the machine)

Now verify that the machine booted properly.

5.7 Build and install models

5.7.1 Create safe.snap backup files for all models

For RCG 2.9, all models will require a safe.snap file. So, before we get started, we should make sure we have such a file for all front-end models, including IOP models.

Typical practice has been to store in the appropriate 'burtfiles' folder in the cds_user_apps repository, with a soft-link in the model's target directory. So we want to create such snapshot files for any models that do not have them

1. Log into a workstation as 'controls'
2. target (to go to the target folder /opt/rtcds/<site>/<ifo>/target)
3. ls –l <ifo>*/<ifo>*/burt/safe.snap
4. Examine the list to find any front-end models that are missing. Likely all the IOP models are missing.
5. For each such missing file, create one using the 'makeSafeBackup' utility. For IOP models, the 'cds' subsystem is chosen. For example, the LSC IOP model would be

```
makeSafeBackup cds l1ioplsc0
```

5.7.2 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

3. Log into boot server as 'controls'

4. cd $RTCDSROOT/chans/ipc
5. mv L1.ipc L1_<date>.ipc
6. touch L1.ipc

5.7.3 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

5.7.4 Install all new models

1. cd to build area.
2. make installWorld

5.8 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

5.9 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)