

Blu-Ice/DCS Administrator's Manual for Release

3.2

Scott McPhillips
Stanford Synchrotron Radiation Laboratory
scottm@slac.stanford.edu

March 11, 2003

Contents

1	Introduction	5
2	Software License Agreement	5
3	Using CVS for source code control	6
3.1	Accessing SSRL's CVS repository	6
3.2	CVS software projects	7
3.3	CVS management policies	8
3.4	CVS naming conventions for tags and branches	9
4	Installation	11
4.1	Installing the libraries	11
4.1.1	Installing the XOS library	11
4.1.2	Installing the auth library	12
4.1.3	Installing the diffimage library	13
4.1.4	Building the jpegsoc library	14
4.1.5	Installing the newmat10 library	14
4.1.6	Building the tcl_clibs library	15
4.1.7	Installing the dcs_tcl_packages library	18
4.1.8	Installing the widgets library	19
4.1.9	Installing the mysql client libraries	20
4.2	Installing DCSS	20
4.3	Creating the mysql database schema	23
4.4	Configuring DCSS	24

4.4.1	Miscellaneous show-stoppers	24
4.4.2	Quick introduction to the database.dat file	24
4.4.3	Configuring DCSS' server ports	25
4.4.4	Configuring the beam line console (NEW for Release 3.2!)	26
4.4.5	Configuring the mysql User permissions tables	26
4.4.6	Starting and Stopping DCSS	28
4.5	Installing and configuring the hardware simulator	29
4.6	Installing and Starting the Diffraction Image Server	30
4.7	Installing BLU-ICE	31
4.8	Configuring BLU-ICE	34
4.9	Installing SSRL's DHS	37
4.10	Configuring SSRL's DHS	39
4.10.1	Miscellaneous show-stoppers for DHS	42
4.10.2	Starting and Stopping DHS	42
4.11	Installing the detector simulator	43
5	Maintaining the database.dat file	44
5.1	Dumping the database.dat file	44
5.2	Recreating the database.dat file	45
5.3	Format of the database.dat file	45
5.3.1	The Real Motor Entry	46
5.3.2	The Pseudo Motor Entry	48
5.3.3	The DHS definition	49
5.3.4	The Ion Chamber Entry	50
5.3.5	The Shutter Entry	50
5.3.6	The Run Definition Entry	51
5.3.7	The Runs Definition entry	52
5.3.8	The Operation Entry	53
5.3.9	The Encoder Entry	54
5.3.10	The String Entry	54
5.3.11	The Device Permissions Bits	54
6	Writing Scripted Devices and Operations	55
6.1	General DCS Scripting Commands	57
6.1.1	Writing to the log window in BLU-ICE	58
6.1.2	Querying a motor position	58
6.1.3	Moving a motor	59
6.1.4	Inserting/Removing shutters and filters	60
6.1.5	Waiting for hardware	60

6.1.6	Using ion chambers	61
6.1.7	Using encoders	62
6.1.8	Using Operations	62
6.1.9	Starting an operation	62
6.1.10	Obtaining operation results	63
6.2	Scripted Device Family Relationships	64
6.2.1	Children and Parents	65
6.2.2	Siblings	66
6.2.3	Observers	67
6.3	Exception Handling	67
6.3.1	The legalities of problems with motors	68
6.3.2	Handling the global abort	69
6.3.3	Handling DHS Crashes	69
6.3.4	Throwing your own exceptions	69
6.3.5	Writing specialized exception handlers	69
7	Adding Scripts to the Scripting Engine	70
7.1	Adding New Scripted Devices	70
7.2	Adding New Scripted Operations	72
8	Example scripts	73
8.1	Testing the diffractometer	73
8.2	The energy device on different beam lines	73
9	The DCS Protocol	76
9.1	The DCS Message Structure	79
9.1.1	The DCS message header	79
9.1.2	The text section	80
9.1.3	The binary section	80
9.2	Connection Protocol	81
9.3	Gui to Server Messages (gtoS)	82
9.3.1	gtoS_abort_all	83
9.3.2	gtoS_become_master	83
9.3.3	gtoS_become_slave	83
9.3.4	gtoS_configure_device	83
9.3.5	gtoS_read_ion_chambers	85
9.3.6	gtoS_set_motor_position	85
9.3.7	gtoS_set_shutter_state	85
9.3.8	gtoS_start_motor_move	86
9.3.9	gtoS_start_oscillation	86

9.3.10	gto_start_operation	87
9.4	Hardware to Server Messages (htos)	87
9.4.1	htos_client_is_hardware	87
9.4.2	htos_configure_device	88
9.4.3	htos_motor_move_completed	88
9.4.4	htos_motor_move_started	89
9.4.5	htos_update_motor_position	89
9.4.6	htos_report_shutter_state	89
9.4.7	htos_report_ion_chambers	90
9.4.8	htos_send_configuration	90
9.4.9	htos_simulating_device	90
9.4.10	htos_operation_update	91
9.4.11	htos_operation_completed	91
9.5	Server To GUI Client Messages (stog)	92
9.5.1	stog_become_master	92
9.5.2	stog_become_slave	92
9.5.3	stog_configure_real_motor	92
9.5.4	stog_configure_pseudo_motor	92
9.5.5	stog_motor_move_completed	92
9.5.6	stog_motor_move_started	92
9.5.7	stog_no_hardware_host	92
9.5.8	stog_other_master	93
9.5.9	stog_report_ion_chambers	93
9.5.10	stog_report_shutter_state	93
9.5.11	stog_simulating_device	93
9.5.12	stog_unrecognized_command	93
9.5.13	stog_update_motor_position	93
9.5.14	stog_operation_completed	93
9.5.15	stog_operation_update	94
9.6	Server To Hardware Messages (stoh)	94
9.6.1	stoh_abort_all	94
9.6.2	stoh_correct_motor_position	94
9.6.3	stoh_start_motor_move	94
9.6.4	stoh_configure_real_motor	94
9.6.5	stoh_configure_pseudo_motor	95
9.6.6	stoh_read_ion_chambers	96
9.6.7	stoh_set_motor_position	96
9.6.8	stoh_set_shutter_state	96
9.6.9	stoh_start_oscillation	96
9.6.10	stoh_start_operation	96

10 Example code listings in ASCII	97
10.0.1 Test Diffractometer BLU-ICE Script	97
10.0.2 Test Diffractometer Operation Script	97
11 Adding new hardware support	97
11.0.1 Example: Adding MAR CCD support	97
12 Document Version Information	100

1 Introduction

This document is an assortment of smaller documents related to the Blu-Ice/DCS software project. It was written to assist software developers and/or beam line administrators in installing and configuring the complete software. It may also be of interest to those that wish to understand what is required in order to get Blu-Ice/DCS up and running on a new beam line.

2 Software License Agreement

Copyright 2001
by
The Board of Trustees of the
Leland Stanford Junior University
All rights reserved.

Disclaimer Notice

The items furnished herewith were developed under the sponsorship of the U.S. Government. Neither the U.S., nor the U.S. D.O.E., nor the Leland Stanford Junior University, nor their employees, makes any warranty, express or implied, or assumes any liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use will not infringe privately-owned rights. Mention of any product, its manufacturer, or suppliers shall not, nor is it intended to, imply approval, disapproval, or fitness for any particular use. The U.S. and the University at all times retain the right to use and disseminate the furnished items for any purpose whatsoever.

Notice 91 02 01

Work supported by the U.S. Department of Energy under contract DE-AC03-76SF00515; and the National Institutes of Health, National Center for Research Resources, grant 2P41RR01209.

Permission Notice

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3 Using CVS for source code control

Currently the DCS/Blu-Ice software is maintained and managed using CVS. The web offers excellent CVS Documentation¹ to help a developer get started. Specifics about how the Blu-Ice/DCS software uses CVS is described more thoroughly in the following sub-sections.

3.1 Accessing SSRL's CVS repository

To access the CVS repository you will need a username and password provided by SSRL. CVS does not provide secure encryption, so the CVS account will not be related to an actual SSRL computer account.

¹<http://cvsbook.red-bean.com/cvsbook.html>

Follow these steps to check out a project from CVS.

1. Log in to CVS. This will talk to the SSRL repository and also write your password onto your local file system so that later CVS commands do not need the password.

```
cvs -d :pserver:yourusername@smb.slac.stanford.edu:/home/code/repository login
Logging in to :pserver:yourusername@smb.slac.stanford.edu:2401/home/code/repository
CVS password:
```

2. Checkout a project, replacing *projectName* in the following statement with an actual project.

```
cvs -d :pserver:yourusername@smb.slac.stanford.edu:/home/code/repository
checkout projectName
```

3. Change into the project directory.

```
cd projectName
```

4. Within the project directory structure, it is not necessary to specify the project's repository when issuing CVS commands. For example, 'cvs diff -r HEAD' will compare your latest version against SSRL's latest version.

Note: In this documentation it will be necessary for external collaborators to change all of the example 'cvs checkout' commands to the full pserver style command:

```
cvs -d :pserver:yourusername@smb.slac.stanford.edu:/home/code/repository
checkout
```

3.2 CVS software projects

The Distributed Control System is composed of various software components or 'projects' within the CVS repository. The projects related to DCS are listed here.

- xos: library of system calls.
- auth: Some simple authentication procedures.
- tcl_clibs: C functions loadable by TCL.
- dcs_tcl_packages: tcl functions used by scripting engine and BLU-ICE.

- dcss: The distributed control system server.
- dhs: SSRL's Distributed hardware server.
- simdhs: A TCL program acting as a DHS, simulating motors and other common hardware.
- widgets: Graphical TCL widgets used by BLU-ICE.
- blu-ice: The GUI client interface to DCSS.
- jpegsoc: Uses jpeg-6b library to send and receive JPEGs over a socket connection.
- diffimage: Loads diffraction images.
- imgsrv: The diffraction image server.

3.3 CVS management policies

Even though the CVS program is a powerful tool for open source development projects, the project itself has a much better chance of succeeding if basic ground rules are provided and consistently used to manage the CVS repository.

1. Only 1 unstable branch

The goal is to support as few branches as possible (branches lead to merges). Developers do not automatically get their own branch to play with. This policy requires that developers communicate and plan out large changes to the code. A mailing list is provided to assist in communication between developers.

2. The HEAD (or trunk) is unstable

The trunk (i.e. HEAD in CVS terminology) will be the unstable version. New features may be submitted only to the HEAD. The HEAD will be considered unstable and fit only for development work. This does not mean that a developer has the right to commit changes in a sloppy fashion.

3. One moderator for each project

There will be a moderator for each software project who is responsible for deciding what software features will be in each stable release.

If a developer wishes to contribute to the repository, but does not have CVS 'commit' access, then the patches should be mailed to this moderator, who will commit the changes as appropriate.

4. Reasonable code standards

The developer should only commit changes that can be built, have been at least minimally tested, and are consistent with the current state of other CVS projects. Changes should also be tested on as many platforms as possible.

Example: Scott wants to change the DCS protocol between Blu-Ice and DCSS. He should not commit any changes to the dcss project unless he is also ready to commit the related changes to the blu-ice project.

5. Stable releases are tagged and branched

Candidate stable versions of the HEAD will be tagged to indicate that it is a release. Bug fixes on stable releases will be handled by branching from this tag. A branch will be made for the tag when a bug fix is available for the stable release.

6. Automatic merging between branches somewhat discouraged

Developers that wish to merge bug fixes that have been applied to a "release branch" to the HEAD are strongly encouraged to do this activity in a manual fashion (e.g. editor?). However, if the CVS merge features have been deemed appropriate for the particular case, the branch can be tagged as described in Section 3.4.

3.4 CVS naming conventions for tags and branches

In general, the software will be labeled with the commonly used *major.minor.patch* notation. For example, a particular software version could be described as DCS version 3.0.1. The specifics for describing software versions using CVS tags is described here:

1. Tagging stable versions

The naming convention for tagging stable versions will use the major and minor numbers. Unfortunately CVS can not handle '.' in tags, so an underscore will be used. The general format will be

`release-major_minor-tag`

The minor number will always be tagged with an even number to avoid confusion with some open source projects that use odd minor numbers to indicate an unstable branch.

2. Tagging the start of a branch

The naming convention for a branch will be the tag convention minus the trailing "tag".

Example:

The branch tag for `release-3_2-tag` is `release-3_2`

3. Tagging a branch for release/export

If a bug fix on a branch needs to be shipped to someone that does not have CVS access, the files in the branch should be tagged by incrementing the patch number. In this way the shipped software can be identified later by looking at the tag.

Example:

Synchrotron X does not have CVS access and needs a bug fix on `release-3_2`. The files are changed on the branch and tagged with a `release-3_2_1`. The files are exported and shipped with the version number.

4. Tagging the branch after a merge from trunk

The naming convention for tagging a branch after it has been merged into the trunk is to increment the patch number followed by a "-merged-date."

Example 1:

A bug fix is applied to `release-3_2` branch. The point comes where the trunk needs this same fix, so the change is merged from the release branch into the trunk with the following command.

```
cvs update -j release-3_2
```

The branch is immediately tagged again to mark this event:

```
cvs tag release-3_2_1-merged-05JUL02
```

Example 2:

If more fixes are applied to the 3.2 branch, it will be possible to again merge the branch into the trunk using this new tag as follows:

```
cvs update -j release-3_2_1-merged-05JUL02 -j release-3_2
```

At this point the branch should again be tagged by incrementing the patch number.

```
cvs tag release-3.2.2-merged-07JUL02
```

A third merge would use the following command:

```
cvs update -j release-3.2.2-merged07JUL02 -j release-3.2.1-05JUL02
```

4 Installation

The first step in installing the DCS software is to create a new directory to install it in. All software packages checked out of SSRL's CVS repository should be checked out into this new directory. This documentation will refer to this directory as the 'DCS root' directory and the examples will use a 'DCS root' directory of `~/release-3_2/`. Most of the build files will build relative to this directory.

Depending on the purpose of a particular computer in the distributed system, it may not be necessary to install all of the DCS software components. For example, it is not necessary to check out the 'blu-ice' onto a computer that will only be used to run a DHS program.

NOTE: This documentation was tested against g++ version 2.96 on Linux Red Hat 7.3.

4.1 Installing the libraries

4.1.1 Installing the XOS library

1. Checkout the xos library into your DCS 'root' directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 xos
cvs checkout: Updating xos
cvs checkout: Updating xos/decunix
U xos/decunix/makefile
U xos/decunix/xos.a
cvs checkout: Updating xos/irix
U xos/irix/makefile
U xos/irix/xos.a
cvs checkout: Updating xos/linux
U xos/linux/makefile
U xos/linux/xos.a
cvs checkout: Updating xos/src
```

```

U xos/src/xos.c
U xos/src/xos.h
U xos/src/xos_hash.c
U xos/src/xos_hash.h
U xos/src/xos_semaphore_set.c
U xos/src/xos_semaphore_set.h
U xos/src/xos_socket.c
U xos/src/xos_socket.h
cvs checkout: Updating xos/win32
U xos/win32/xos.dsp
U xos/win32/xos.dsw

```

2. cd into the linux, decunix, or irix build directory and build the software with the make command.

```

blctlxx:~/release-3_2 > cd xos/linux/
blctlxx:~/release-3_2/xos/linux > make
cc -c -DLINUX -DXOS_PRODUCTION_CODE ../src/ ../src//xos.c
cc: ../src/: linker input file unused since linking not done
cc -c -DLINUX -DXOS_PRODUCTION_CODE ../src/ ../src//xos_hash.c
cc: ../src/: linker input file unused since linking not done
cc -c -DLINUX -DXOS_PRODUCTION_CODE ../src/ ../src//xos_semaphore_set.c
cc: ../src/: linker input file unused since linking not done
cc -c -DLINUX -DXOS_PRODUCTION_CODE ../src/ ../src//xos_socket.c
cc: ../src/: linker input file unused since linking not done
ar -vr xos.a xos.o xos_hash.o xos_semaphore_set.o xos_socket.o
r - xos.o
r - xos_hash.o
r - xos_semaphore_set.o
r - xos_socket.o

```

4.1.2 Installing the auth library

1. Checkout the auth software into your DCS 'root' directory.

```

blctlxx:~/release-3_2 > cvs checkout -r release-3_2 auth
cvs checkout: Updating auth
cvs checkout: Updating auth/decunix
U auth/decunix/auth.a
U auth/decunix/makefile

```

```

cvs checkout: Updating auth/irix
U auth/irix/auth.a
U auth/irix/makefile
cvs checkout: Updating auth/linux
U auth/linux/auth.a
U auth/linux/makefile
cvs checkout: Updating auth/src
U auth/src/auth.c
U auth/src/auth.h

```

2. cd into the linux, decunix, or irix build directory and build the software with the make command.

```

blctlxx:~/release-3_2 > cd auth/linux/
blctlxx:~/release-3_2/auth/linux > make
cc -c -O -DLINUX -I../ -I../xos/src/ ../src/auth.c
../src/auth.c: In function 'auth_load_key':
../src/auth.c:426: warning: assignment makes integer from pointer without a cast
ar -vr auth.a auth.o
r - auth.o

```

4.1.3 Installing the diffimage library

1. Checkout the diffimage software into your DCS 'root' directory.

```

blctlxx:~/release-3_2 > cvs checkout -r release-3_2 diffimage
cvs checkout: Updating diffimage
cvs checkout: Updating diffimage/decunix
U diffimage/decunix/makefile
cvs checkout: Updating diffimage/irix
U diffimage/irix/diffimage.a
U diffimage/irix/makefile
cvs checkout: Updating diffimage/linux
U diffimage/linux/makefile
cvs checkout: Updating diffimage/src
U diffimage/src/diffimage.c
U diffimage/src/diffimage.h
U diffimage/src/libimage.c
U diffimage/src/libimage.h
U diffimage/src/marheader.c
U diffimage/src/marheader.h

```

2. cd into the `linux`, `decunix`, or `irix` build directory and build the software with the `make` command.

4.1.4 Building the jpegsoc library

1. Make sure that you have the jpeg-6b library installed on your computer.
2. Checkout the jpegsoc software into the your DCS 'root' directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 jpegsoc
cvs checkout: Updating jpegsoc
cvs checkout: Updating jpegsoc/decunix
U jpegsoc/decunix/makefile
cvs checkout: Updating jpegsoc/irix
U jpegsoc/irix/makefile
cvs checkout: Updating jpegsoc/linux
U jpegsoc/linux/makefile
cvs checkout: Updating jpegsoc/src
U jpegsoc/src/jdatadstsock.c
U jpegsoc/src/jdatasrcsock.c
U jpegsoc/src/jpegsoc.h
U jpegsoc/src/jsndrcv.c
```

3. cd into the `linux`, `decunix`, or `irix` build directory and build the software with the `make` command.

```
blctlxx:~/release-3_2 > cd jpegsoc/linux/
blctlxx:~/release-3_2/jpegsoc/linux > make
gcc -c -O -DLINUX -DAPPLACK -I../xos/src/ -I../src -I/usr/local/ -I/usr/
gcc -c -o jdatasrcsock_applack.o -O -DLINUX -DAPPLACK -I../xos/src/ -I..
a - jsndrcv.o
a - jdatasrcsock_applack.o
a - jdatadstsock_applack.o
```

4.1.5 Installing the newmat10 library

1. Create a directory, perhaps in your DCS 'root' directory (e.g. `~/release-3_2/newmat10/linux`) which can be referenced later by the `tcl_clibs` makefile.

2. Download the `newmat10.tar.gz`² file into this new directory.
3. `gunzip newmat10.tar.gz`
4. `tar -xvf newmat10.tar`
5. Use the correct makefile for your computer architecture. For example, on Linux:

```
blctlxx:~/release-3_2/newmat10/linux > make -f nm_gnu.mak
```

Note: There have been reports of build problems using gcc 3.2. If this is the case for you, then you may go on to the next step and modify the `tcl_clibs` to build without this library. This library is only used to calculate optimal table position for SSRL beam lines, and you may not need this feature.

4.1.6 Building the `tcl_clibs` library

This is a library of C functions that a TCL program is able to load at runtime. This library is loaded both by the Blu-Ice program and by the DCSS scripting engine.

1. Install as described in Section 4.1.5.
2. Install the `jpeg` library as described in Section 4.1.4.
3. Checkout the `tcl_clibs` software into your DCS 'root' directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 tcl_clibs
cvs checkout: Updating tcl_clibs
cvs checkout: Updating tcl_clibs/decunix
U tcl_clibs/decunix/makefile
cvs checkout: Updating tcl_clibs/irix
U tcl_clibs/irix/makefile
U tcl_clibs/irix/tcl_clibs.so
cvs checkout: Updating tcl_clibs/linux
U tcl_clibs/linux/calibrate.so
U tcl_clibs/linux/makefile
cvs checkout: Updating tcl_clibs/src
```

²<http://www.robertnz.net/download.html>

```

U tcl_clibs/src/analyzePeak.c
U tcl_clibs/src/analyzePeak.h
U tcl_clibs/src/calibrate.c
U tcl_clibs/src/calibrate.h
U tcl_clibs/src/cubgcv.c
U tcl_clibs/src/cubgcv.h
U tcl_clibs/src/ice.c
U tcl_clibs/src/ice_auth.c
U tcl_clibs/src/ice_auth.h
U tcl_clibs/src/ice_cal.c
U tcl_clibs/src/ice_cal.h
U tcl_clibs/src/image_channel.c
U tcl_clibs/src/image_channel.h
U tcl_clibs/src/tcl_macros.h

```

4. cd into the linux, decunix, or irix build directory and build the software with the make command.

```

blctlxx:~/release-3_2 > cd tcl_clibs/linux/
blctlxx:~/release-3_2/tcl_clibs/linux > make
g++ -c -O -DLINUX -Wall -I./ -I../ -I/usr/local/include/ -I../..../auth/src -
g++ -c -O -DLINUX -Wall -I./ -I../ -I/usr/local/include/ -I../..../auth/src -
g++ -c -I../..../newmat10/linux/ ../src/calibrate.c
../src/calibrate.c: In function 'int cal_convolutegaussian
(ColumnVector &, ColumnVector &, double)':
../src/calibrate.c:205: warning: assignment to 'int' from 'double'
../src/calibrate.c:205: warning: argument to 'int' from 'double'
g++ -c -O -DLINUX -Wall -I./ -I../ -I/usr/local/include/ -I../..../auth/src -
g++ -c -O -DLINUX -Wall -I./ -I../ -I/usr/local/include/ -I../..../auth/src -
../xos/src/ -I/usr/lib/ -I../..../jpegsoc/src ../src/image_channel.c
g++ -c -O -DLINUX -Wall -I./ -I../ -I/usr/local/include/ -I../..../auth/src -
../xos/src/ -I/usr/lib/ -I../..../jpegsoc/src ../src/analyzePeak.c
../src/analyzePeak.c: In function 'int nonzero (double *, int)':
../src/analyzePeak.c:336: warning: 'int nzero' might be used
uninitialized in this function
../src/analyzePeak.c: In function 'int find_extrema (int, double *,
double *, double **, double *, double *, int *, double *, double *, int
*)':
../src/analyzePeak.c:348: warning: 'double root1' might be used
uninitialized in this function

```

```

./src/analyzePeak.c:348: warning: 'double ex1' might be used
uninitialized in this function
./src/analyzePeak.c:348: warning: 'double ex2' might be used
uninitialized in this function
g++ -c -O -DLINUX -Wall -I./ -I../ -I/usr/local/include/ -I../auth/src -I.
../xos/src/ -I/usr/lib/ -I../jpegsoc/src ./src/cubgcv.c
./src/cubgcv.c: In function 'int cubgcv (double *, double *, double *,
int, double *, double **, int, double *, int, double *, double **,
double **, double *, double *, int *)':
./src/cubgcv.c:258: warning: label 'L150' defined but not used
./src/cubgcv.c:253: warning: label 'L140' defined but not used
g++ -shared ice.o ice_cal.o calibrate.o ice_auth.o image_channel.o analyzePea
k.o cubgcv.o ../../jpegsoc/linux/jpegsoc_applack.a ../../newmat10/linux/libnew
mat.a ../../auth/linux/auth.a ../../xos/linux/xos.a /usr/lib/libjpeg.a -lc /
usr/lib/libtcl8.3.so -lm -lpthread -o tcl_clibs.so

```

5. Test the library by loading it from the tcl prompt and manually calling a provided procedure. (This also tests the newmat10 library build.)

```

smb1x5:~> wish
% load ~/release-3_2/tcl_clibs/linux/tcl_clibs.so dcs_c_library
% cal_find_peak 5 "1 2 3 4 5" "1 2 3 2 1"
3.000000e+00 3.000000e+00
% exit

```

IMPORTANT: Make the 'load' command from the wish command prompt work before you attempt to install blu-ice and dcsc. Its much easier to figure out why the load command doesn't work at this stage than it is to figure out what is wrong at the start-up of blu-ice.

Note: If you were unsuccessful in build newmat10, this library will not build. You can attempt to remove the references to the newmat10 library by changing the makefile and ice.c file as shown in this diff output:

```

smb1x5:~/release-3_2/tcl_clibs > cvs diff
cvs diff: Diffing .
cvs diff: Diffing decunix
cvs diff: Diffing irix
cvs diff: Diffing linux
Index: linux/makefile
=====

```

```

RCS file: /home/code/repository/tcl_clibs/linux/makefile,v
retrieving revision 1.8
diff -r1.8 makefile
65c65
< LOCAL_LIBS = ${JPEG_EXT_DIR}linux/jpegsoc_applack.a ${NEWMAT_LIB_DIR}libnewmat
---
> LOCAL_LIBS = ${JPEG_EXT_DIR}linux/jpegsoc_applack.a ${AUTH_LIB_DIR}auth.a ${
70c70
< OBJS = ice.o ice_cal.o calibrate.o ice_auth.o image_channel.o analyzePeak.o cu
---
> OBJS = ice.o ice_auth.o image_channel.o analyzePeak.o cubgcv.o
85,90d84
<
< ice_cal.o : ${CODE}ice_cal.c
< ${CC} -c ${CFLAGS} ${CPP_CONSTANTS} ${INCLUDE_DIR} ${CODE}ice_cal.c
<
< calibrate.o : ${CODE}calibrate.c
< ${CC} -c -I${NEWMAT_DIR} ${CODE}calibrate.c
cvs diff: Diffing src
Index: src/ice.c
=====
RCS file: /home/code/repository/tcl_clibs/src/ice.c,v
retrieving revision 1.11
diff -r1.11 ice.c
51,52d50
< REGISTER_TCL_COMMAND( cal_find_peak );
< REGISTER_TCL_COMMAND( cal_correct_energy );

```

4.1.7 Installing the dcs_tcl_packages library

1. Checkout dcs_tcl_packages from the CVS repository into your DCS 'root' directory.

```

blctlxx:~/release-3_2 > cvs checkout -r release-3_2 dcs_tcl_packages
cvs checkout: Updating dcs_tcl_packages
U dcs_tcl_packages/DcsNetworkProtocol.tcl
U dcs_tcl_packages/DcsRunSequenceViewer.tcl
U dcs_tcl_packages/DcsRunSequencer.tcl

```

4.1.8 Installing the widgets library

1. Checkout widgets from the CVS repository into your DCS 'root' directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 widgets
cvs checkout widgets
cvs checkout: Updating widgets
U widgets/BIWCif.tcl
U widgets/BIWComponent.tcl
U widgets/BIWEntry.tcl
U widgets/BIWGraph.tcl
U widgets/BIWLabeledFrame.tcl
U widgets/BIWMDI.tcl
U widgets/BIWMenu.tcl
U widgets/BIWResolution.tcl
U widgets/BIWSet.tcl
U widgets/BIWUtil.tcl
U widgets/BIWVideo.tcl
U widgets/DCSDevice.tcl
U widgets/DCSDeviceView.tcl
U widgets/pkgIndex.tcl
cvs checkout: Updating widgets/Cif
cvs checkout: Updating widgets/Graph
cvs checkout: Updating widgets/Graph/demo
U widgets/Graph/demo/sampleFile.cif
cvs checkout: Updating widgets/MDI
cvs checkout: Updating widgets/MDI/bitmaps
cvs checkout: Updating widgets/MDI/demo
cvs checkout: Updating widgets/Menu
cvs checkout: Updating widgets/Menu/demo
cvs checkout: Updating widgets/Util
cvs checkout: Updating widgets/demo
U widgets/demo/DeviceDemo.tcl
U widgets/demo/EntryDemo.tcl
U widgets/demo/GraphApp.tcl
U widgets/demo/GraphDemo.tcl
U widgets/demo/LabeledFrameDemo.tcl
U widgets/demo/MDI_test.tcl
U widgets/demo/MenuTest.tcl
```

```
U widgets/demo/ResolutionWidgetApp.tcl
U widgets/demo/sampleFile.bip
```

4.1.9 Installing the mysql client libraries

DCSS and the standard DHS both need the mysql client libraries installed in order to build correctly. The latest versions of Red Hat Linux usually have the client libraries installed by default. Other computer architectures may need to install this library.

4.2 Installing DCSS

Each beam line must have one (and only one) installation of DCSS. The DCSS program must run on a computer that has network access to the computers that will be running the BLU-ICE clients as well as the computers running the Distributed Hardware Servers (DHS's). At SSRL, the DHS's usually run on computers that are on a private network, whereas the BLU-ICE's clients run on computers that are on a more open public network. This architecture implies that the DCSS program must run on a computer that is multi-homed and able to see both the public and private networks.

1. Verify that you have a recent version of TCL installed on your computer.

```
blctlxx:~/release-3_2 > tclsh
% info tclversion
8.3
% exit
blctlxx:~/release-3_2 >
```

2. Install the mysql client library as described in Section 4.1.9.
3. Install the xos library as described in Section 4.1.1.
4. Install the auth library as described in Section 4.1.2.
5. Install the tcl.clibs library as described in Section 4.1.6.
6. Install the dcs_tcl_packages library as described in Section 4.1.7.
7. Checkout the DCSS software into your DCS 'root' directory

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 dcss
```

8. In 'src/dcss_users.c', there is a hardcoded reference to the 'smb' host database. Change this smb to the hostname of the machine running your mysql database.

For example, if to change the database to local host you should expect the following diff results:

```
blctlxx:~/release-3_2/dcss/src > cvs diff
cvs diff: Diffing .
Index: dcss_users.c
=====
RCS file: /home/code/repository/dcss/src/dcss_users.c,v
retrieving revision 1.8
diff -r1.8 dcss_users.c
94c94
<  "smb","dcss",0,"beamline_configuration",0,0,0);
---
>  "localhost","dcss",0,"beamline_configuration",0,0,0);
blctlxx:~/release-3_2/dcss/src >
```

9. If you are building DCSS on a non-linux computer, you may need to change the 'scripts/engine/scriptingEngine.tcl' file to reference the correct build directory for the tcl_clibs.so file.

```
blctlxx:~/release-3_2/dcss/scripts/engine > grep tcl_clibs *
scriptingEngine.tcl:load $DCS_DIR/tcl_clibs/linux/tcl_clibs.so dcs_c_library
%$
```

10. cd into the linux or decunix directory.

```
blctlxx:~/release-3_2/ > cd dcss/linux/
```

11. Build the software with the make command.

```
blctlxx:~/release-3_2/dcss/linux > make
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I../../xos/
../src/dcss_main.c: In function 'start_server':
../src/dcss_main.c:205: warning: passing arg 2 of 'xos_thread_create' from incompatib
```

```

../src/dcss_main.c:217: warning: passing arg 2 of 'xos_thread_create' from i
../src/dcss_main.c:228: warning: passing arg 2 of 'xos_thread_create' from i
../src/dcss_main.c: In function 'restore_database':
../src/dcss_main.c:290: warning: int format, device_type_t arg (arg 3)
../src/dcss_main.c:323: warning: int format, circle_mode_t arg (arg 15)
../src/dcss_main.c:386: warning: int format, circle_mode_t arg (arg 9)
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
../src/dcss_scripting.c: In function 'scripting_thread':
../src/dcss_scripting.c:50: warning: implicit declaration of function 'Itcl_
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
../src/dcss_gui_client.c: In function 'gtos_start_motor_move':
../src/dcss_gui_client.c:1101: warning: unused variable 'correction'
../src/dcss_gui_client.c: In function 'gtos_start_oscillation':
../src/dcss_gui_client.c:1293: warning: unused variable 'correction'
../src/dcss_gui_client.c: In function 'gtos_start_vector_move':
../src/dcss_gui_client.c:1335: warning: unused variable 'device_2'
../src/dcss_gui_client.c:1333: warning: 'deviceNum_2' might be used uninitia
../src/dcss_gui_client.c:1338: warning: 'device2isNull' might be used uninit
../src/dcss_gui_client.c: In function 'gtos_stop_vector_move':
../src/dcss_gui_client.c:1414: warning: unused variable 'device_2'
../src/dcss_gui_client.c: In function 'gtos_change_vector_speed':
../src/dcss_gui_client.c:1479: warning: unused variable 'device_2'
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
../src/dcss_hardware_client.c: In function 'write_to_hardware':
../src/dcss_hardware_client.c:862: warning: 'result' might be used uninitial
cc: -lsocket: linker input file unused since linking not done
cc -c -pthread -O5 -lsocket -DLINUX -DSEC_BASE -DSEC_NET -Wall -I../src -I.
cc: -lsocket: linker input file unused since linking not done
cc -o dcss -lz dcss_main.o dcss_database.o dcss_client.o dcss_scripting.o dc

```

4.3 Creating the mysql database schema

The DCSS code uses several tables in the mysql database to determine who is currently allowed to use a beam line.

The standard SSRL DHS code also uses the same database to extract device configurations at start up.

To create this database, first install dcss as described in Section 4.2, which contains three example database dumps in the the `/dcss/examples/` directory.

- `databaseTables.txt`

This file is the schema only for the database. It does not contain any example entries.

- `exampleFullDatabase.txt`

This file is the same as `databaseTables.txt` except that the file also contains a snapshot of the state of SSRL's database on a particular date. This file is probably too cluttered for everyday use.

- `simpleTables.txt`

This file is the same as the `databaseTables.txt` except that it contains a few simple entries for a simple beam line. This is probably a good starting point for most developers.

The following directions are provided which will assist in creating the database schema.

1. Log in to the mysql database.

```
blctlxx:~/release-3_2/dcss/examples >mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12313 to server version: 3.23.53a

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

2. Create an empty database called `beamline_configuration`.

```
mysql> CREATE DATABASE beamline_configuration;
```

3. From the unix shell prompt type:

```
mysql -u root -p beamline_configuration < simpleTables.txt
```

4. At some point be sure to grant `select` permissions for `dcss` and all `dhs` programs and related hostnames. See the instructions regarding the 'GRANT' command in Section 4.4.5 and Section 4.10.

4.4 Configuring DCSS

DCSS is the core of the DCS system. The 'dcss/src' directory contains 'C' code which acts mostly as a message router. This code also maintains a memory mapped file (i.e. the `database.dat` file), where copies of the motor positions and general configuration of the beam line devices are stored.

The 'dcss/scripts/engine' directory is where the TCL 'scripting engine' code is located. The 'dcss/src' and 'dcss/scripts/engine' directories are core code and modifying this code could fundamentally change the behavior of the system.

The 'dcss/scripts/devices' directory is where the 'scripted device' definitions are described in tcl scripts. The 'dcss/scripts/operations' directory is where the scripts for automation and complex sequencing are placed.

The following subsections describe the steps for configuring a simple beam line.

4.4.1 Miscellaneous show-stoppers

1. Create a directory `/home/adm/keys` which is world writable.
2. The "SELECT" privileges must be enabled for `dcss` in the `mysql` database as described in Section 4.4.5.

4.4.2 Quick introduction to the `database.dat` file

After installing the DCSS program, it is necessary to create a valid `database.dat` file. Refer to Section 5 for a full reference on how to create and use this file. If you are just getting started and want to play with a simulation, an example database is available in text format. To use this file, follow the these steps:

1. Copy the example dump file into the directory with the DCSS binary.

```
blctlxx:~/release-3_2/dcss/linux > cp ../examples/example_b1111sim_dump.txt
```
2. Using a text editor, edit the DHS entries to match the hostnames of the computers running DHS's on the new beam line. For example, the third line of a DHS entry may need to change to reflect the name of

a DHS running on a different computer. If you plan to run the DHS on the same machine as DCSS, then you can use `localhost` as the hostname.

Example:

```
simDhs
3
localhost 2
```

3. Create the `database.dat` file with the `./dcss -r` command.

Note: This step will completely overwrite an existing `database.dat` file, which usually contains current motor positions.

```
blctlxx:~/release-3_2/dcss/linux > ./dcss -r example_bl1111sim_dump.txt
....<OUTPUT REMOVED>.....
```

A total of 171 devices were read in from the dump file.

4.4.3 Configuring DCSS' server ports

4. Copy the `serverPorts.txt` file from the `examples` directory into the top `'dcss'` directory.

```
blctlxx:~/release-3_2/dcss > cp examples/serverPorts.txt .
```

The `serverPorts.txt` file is read by DCSS at start-up to determine what the listening ports are for the clients. The format is simply a single line of text:

```
scriptingEnginePort hardwarePort guiClientPort
```

where *scriptingEnginePort* is the port number which the scripting engine connects to. The *hardwarePort* for all of the hardware clients (DHS's) to connect to. The *guiClientPort* is the port number for which all of the gui clients connect to.

NOTE: The contents of this file could be added to the mysql database in a future release.

4.4.4 Configuring the beam line console (NEW for Release 3.2!)

DCSS uses the blu-ice's host DISPLAY variable and restricts a user's capability based on this variable and the state of the hutch door.

- (a) Copy the `displays.txt` file from the examples directory into the top 'dcss' directory.

```
blctlxx:~/release-3_2/dcss > cp examples/displays.txt .
```

- (b) Edit the file and classify your machines' consoles.

The `displays.txt` file is read by DCSS once at start-up and enables DCSS to determine whether or not a Blu-Ice client is 'local', in the 'hutch', or at a 'remote' location. The format is list of entries, one line for each possible display that DCSS should know about. If a Blu-Ice logs in from a display that is not listed in this file, the Blu-Ice will automatically be categorized as 'remote'.

The format of each line entry is as follows:

category hostname display [#Description]

- The *category* can be either 'hutch', 'local', or 'remote'.
- The *hostname* is the X-windows HOST environment variable for the machine that Blu-Ice is displaying on. This field is read but not used if the *display* variable starts with something other than a ':'.
• The *display* is the X-windows DISPLAY variable for the console that Blu-Ice is displaying on. If the display starts with a ':', then the *hostname* is used to determine the Blu-Ice client's location.
- The line may optionally add a comment describing the location of the terminal. This comment is broadcast to all of the Blu-Ice clients, further clarifying the location of the GUI.

NOTE: This file could be added to the mysql database schema at some point.

4.4.5 Configuring the mysql User permissions tables

DCSS uses the mysql database to determine which users are allowed to user the beam line.

- (a) Configure and/or install the mysql database as discussed in Section 4.3.
- (b) Make sure that the mysql database will allow the new machine to do an SQL "SELECT" from the appropriate tables in the beamline_configuration database:

```
bl921:~ > /usr/local/mysql/bin/mysql -u admin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 354856 to server version: 3.22.32
```

Type 'help' for help.

```
mysql> use beamline_configuration
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
mysql> grant select on DCSS to dcss@blctlxx.slac.stanford.edu;
mysql> grant select on Users to dcss@blctlxx.slac.stanford.edu;
mysql> grant select on Beamlines to dcss@blctlxx.slac.stanford.edu;
Query OK, 0 rows affected (0.17 sec)
```

```
mysql> flush privileges;
Query OK, 0 rows affected (0.55 sec)
```

```
mysql>
```

- (c) Add an entry in the mysql database in the Beamlines table with the new beam line name. The id should be used in the next when adding the BeamlineID.

```
mysql> select * from Beamlines where id=12;
+----+-----+
| id | Name |
+----+-----+
| 12 | bl-xx |
+----+-----+
1 row in set (0.02 sec)
```

- (d) Add an entry to the mysql database in the DCSS table. If this is

the 12th beam line defined, then the result may look something like this. The DHS can use the ListeningPort and HostName of this table to find out where its DCSS is located.

```
mysql> select * from DCSS where id=12;
+----+-----+-----+-----+-----+
| id | HostName                | ListeningPort | MemoryMapFile | Beamline |
+----+-----+-----+-----+-----+
| 12 | blctlxx.slac.stanford.edu |          14242 | database.dat   |          |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Note: The ListeningPort entry in this not used by DCSS, because DCSS gets its port assignments from the 'serverPorts.txt' as described in as described in Section 4.4.3. However, the standard SSRL DHS actually does look into this table and grabs this value to determine which host and port it should connect to. Therefore, this Listening port should be the same value as the 'hardware-Port' as described in the serverPorts.txt file. Obviously there is some clean up needed here.

- (e) Add an entry to the Users table for your account. The `beamline` field should either match the name of the `beamline` field in the DCSS table or the entry could be `ALL`, which would give the user access to all of the defined beamlines. The `priv` field is not used anymore and will be deleted in future release.

```
mysql> select * from Users;
+----+-----+-----+-----+-----+-----+
| id | unix_name | priv | real_name                | office_phone | job_title |
+----+-----+-----+-----+-----+-----+
| 1  | scottm   | 4    | Scott McPhillips        | 9-714-xxxx  | S/W Develop |
+----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

4.4.6 Starting and Stopping DCSS

A fully configured dcsc should be started in the dcsc build directory.

```
blctlxx:~/release-3_2/dcsc/linux > ./dcsc -s
```

NOTE: You will need to run DCSS as root if you intend to authenticate users with accounts other than the account running dcsc.

There are no special procedures for stopping DCSS. Control-c at the terminal or the unix 'kill' command are the recommended ways of stopping dcss. Of course it is advisable to first make sure that there are no moving motors before shutting down DCSS.

4.5 Installing and configuring the hardware simulator

The hardware simulator is a TCL script which will connect to DCSS and announce itself as the 'simdhs' hardware server. All entries in the DCCS `database.dat` file that are controlled by the 'simdhs' hardware server will be simulated.

The script will simulate motors, shutters, and ion chambers.

1. Checkout the hardware simulator from CVS into your DCS root directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 simdhs
cvs checkout: Updating simdhs
cvs checkout: Updating simdhs/examples
U simdhs/examples/serverPorts.txt
cvs checkout: Updating simdhs/scripts
U simdhs/scripts/DcssConnection.tcl
U simdhs/scripts/ionChambersClass.tcl
U simdhs/scripts/message_handlers.tcl
U simdhs/scripts/motorClass.tcl
U simdhs/scripts/operations.tcl
U simdhs/scripts/set.tcl
U simdhs/scripts/simdhs.tcl
U simdhs/scripts/util.tcl
```

2. Copy the `simdhs/examples/serverPorts.txt` file into the top directory of the `simdhs` directory from the `examples` directory.

```
blctlxx:~/release-3_2/simdhs > cp examples/serverPorts.txt .
```

This file simply contains a single number that tells the script which port it should try to connect to DCSS on. This number should match the 2nd number in the file by the same name in the `/dcss` directory.

3. Verify that the simulator will try to connect to the correct beam line computer.

The statement 'set serverName localhost' in `simdhs/scripts/simdhs.tcl` should be changed to point at the machine that the DCSS is running on (if it is not running on the the same machine).

4. Start the hardware simulator script in `simdhs/scripts` directory.

```
blctlxx:~/release-3_2/simdhs/scripts > ./simdhs.tcl
```

4.6 Installing and Starting the Diffraction Image Server

1. Checkout the `imgsrv` software into your DCS 'root' directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 imgsrv
cvs checkout: Updating imgsrv
cvs checkout: Updating imgsrv/decunix
U imgsrv/decunix/imgsrv
U imgsrv/decunix/makefile
cvs checkout: Updating imgsrv/irix
U imgsrv/irix/makefile
cvs checkout: Updating imgsrv/linux
U imgsrv/linux/makefile
cvs checkout: Updating imgsrv/src
U imgsrv/src/filebrowser.c
U imgsrv/src/imgsrv_cache.c
U imgsrv/src/imgsrv_cache.h
U imgsrv/src/imgsrv_client.c
U imgsrv/src/imgsrv_client.h
U imgsrv/src/imgsrv_main.c
```

2. `cd` into the `linux`, `decunix`, or `irix` build directory and build the software with the `make` command.

```
blctlxx:~/release-3_2 > cd imgsrv/linux/
blctlxx:~/release-3_2/imgsrv/linux > make
```

3. Start the Diffraction Image Server in the build directory:

```
blctlxx:~/release-3_2/imgsrv/linux > ./imgsrv guiPort webPort
tempImageDirectory
```

where

- *guiPort* is the port number which Blu-Ice will connect to request JPEG images
- *webPort* is the port for the web based software to request images
- *tempImageDirectory* is the directory in which the web based software can write temporary JPEG images.

4.7 Installing BLU-ICE

1. Install the SSRL libraries.

- Install the xos library as described in Section 4.1.1.
- Install the auth library as described in Section 4.1.2.
- Install the jpegsoc library as described in Section 4.1.4.
- Install the tcl_clibs library as described in Section 4.1.6.
- Install the widgets library as described in Section 4.1.8.

2. Install all of the required third party packages and libraries.

- Verify that you have a recent version of TCL installed on your computer.

```
tclsh
% info tclversion
8.3
% exit
```

- Verify that the third party library Itcl³ is installed correctly on your computer.

```
bl92a:~ > wish
% package require Itcl
3.2
% exit
```

- Verify that the third party TCL library BWidget⁴ is installed correctly on your computer.

³<http://sourceforge.net/projects/incrtcl/>

⁴<http://sourceforge.net/projects/tcllib/>

```
b192a:~ > wish
% package require BWidget
1.2.1
% exit
b192a:~ >
```

- Verify that the third party TCL library BLT⁵ is installed correctly on your computer.

```
b192a:~ > wish
% package require BLT
2.4
% exit
b192a:~ >
```

- Verify that the third party TCL library Img⁶ is installed correctly on your computer.

```
b192a:~ > wish
% package require Img
1.2.4
% exit
b192a:~ >
```

3. Checkout the blu-ice project from CVS in to your DCS root directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 blu-ice
cvs checkout: Updating blu-ice
cvs checkout: Updating blu-ice/data
U blu-ice/data/calibration_edges.dat
U blu-ice/data/periodic_bl11.dat
U blu-ice/data/periodic_bl15.dat
U blu-ice/data/periodic_bl91.dat
U blu-ice/data/periodic_bl92.dat
cvs checkout: Updating blu-ice/images
U blu-ice/images/beamstop.gif
U blu-ice/images/frontend.gif
U blu-ice/images/frontend_small.gif
U blu-ice/images/gonio.gif
```

⁵<http://sourceforge.net/projects/blt/>

⁶<http://www.xs4all.nl/nijtmans/img.html>

U blu-ice/images/mar165.gif
U blu-ice/images/mar_small.gif
U blu-ice/images/q4_small.gif
U blu-ice/images/splash.gif
U blu-ice/images/splash.jpg
cvs checkout: Updating blu-ice/irix
cvs checkout: Updating blu-ice/scripts
U blu-ice/scripts/Diffimage.tcl
U blu-ice/scripts/SafeEntry.tcl
U blu-ice/scripts/Sequence.tcl
U blu-ice/scripts/SequenceActions.tcl
U blu-ice/scripts/SequenceCrystals.tcl
U blu-ice/scripts/SequenceResultList.tcl
U blu-ice/scripts/SequenceView.tcl
U blu-ice/scripts/beamline_gui.tcl
U blu-ice/scripts/bl11_specific.tcl
U blu-ice/scripts/bl15_specific.tcl
U blu-ice/scripts/bl71_specific.tcl
U blu-ice/scripts/bl91_specific.tcl
U blu-ice/scripts/bl92_specific.tcl
U blu-ice/scripts/bl92sim_specific.tcl
U blu-ice/scripts/blcommon.tcl
U blu-ice/scripts/centering.tcl
U blu-ice/scripts/classify_field.tcl
U blu-ice/scripts/configure_motor.tcl
U blu-ice/scripts/cursors.tcl
U blu-ice/scripts/default_rc.tcl
U blu-ice/scripts/define_scan.tcl
U blu-ice/scripts/dice_tabs.tcl
U blu-ice/scripts/filters.tcl
U blu-ice/scripts/genkey.sh
U blu-ice/scripts/hardware_commands.tcl
U blu-ice/scripts/hutch.tcl
U blu-ice/scripts/hutch_setup.tcl
U blu-ice/scripts/ice.tcl
U blu-ice/scripts/mdw_document.tcl
U blu-ice/scripts/mega_widget.tcl
U blu-ice/scripts/message_handlers.tcl
U blu-ice/scripts/motor_control.tcl
U blu-ice/scripts/plot.tcl

```
U blu-ice/scripts/reswidget.tcl
U blu-ice/scripts/robot_config.tcl
U blu-ice/scripts/scan.tcl
U blu-ice/scripts/scanlog.tcl
U blu-ice/scripts/set.tcl
U blu-ice/scripts/splash.tcl
U blu-ice/scripts/traces.tcl
U blu-ice/scripts/typed_command.tcl
U blu-ice/scripts/user_scan.tcl
U blu-ice/scripts/util.tcl
U blu-ice/scripts/wedge.tcl
cvs checkout: Updating blu-ice/src
```

4.8 Configuring BLU-ICE

1. Modify `blu-ice/scripts/ice.tcl` to reflect your installation.
 - Verify that the first line of the `blu-ice/scripts/ice.tcl` file points to the correct location of the `wish` program. This file is configured to run on a IRIX machine, which stores this file in a different location than a Red Hat Linux installation.

```
blctlxx:~/release-3_2/blu-ice/scripts > which wish
/usr/bin/wish
```

Example:

```
< #!/usr/local/bin/wish
---
> #!/usr/bin/wish
```

- Change the `DCS_DIR` variable to point to your DCS root directory. Production level software at SSRL runs out of the `/usr/local/dcs/` directory.

```
< set DCS_DIR "/usr/local/dcs/"
---
> set DCS_DIR "/home/scottm/release-3_2/"
```

- Modify the location of the `tcl_clibs.so` file to point at the appropriate build directory. By default, it is pointing at the `irix` build directory.

```
< load $DCS_DIR/tcl_clibs/irix/tcl_clibs.so dcs_c_library
---
> load $DCS_DIR/tcl_clibs/linux/tcl_clibs.so dcs_c_library
```

- Modify `ice.tcl` to connect to your DCSS server.

The Blu-Ice program is started with 1 argument – the name of the beam line that it needs to connect to. This argument is used within a large `switch` statement to initialize the Blu-Ice configuration.

You will need to modify this `switch` statement either by changing an existing beam line configuration or by adding a new configuration. Some important parameters within this initialization are:

- `gBeamline(serverName)` hostname location of the DCSS server.
- `gBeamline(serverPort)` port on which Blu-Ice should attempt to connect to DCSS. This should correspond to the third port number entry in the `serverPorts.txt` file in DCSS's configuration.
- `gBeamline(title)` sets the text of the window's frame at the top of the Blu-Ice.
- `gBeamline(beamlineId)` used in various `switch` statements through `blu-ice`.
- `gBeamline(simulation)` changes the color of Blu-Ice only.
- `gBeamline(detector)` lets Blu-Ice display appropriate detector modes and pictures
- `gBeamline(videoServerUrl)` various web address of Axis camera servers.
- `gBeamline(hutchVideoPath)`
- `gBeamline(sampleVideoPath)`
- `gBeamline(ptzPath)` web location of pan-tilt-zoom preset information

2. In `blu-ice/data/` copy one of the existing files to the same directory using a new name:

```
periodic_beamlineID.dat
```

where `beamlineID.dat` is defined by the `gBeamline(beamlineId)` variable. Modify the new file to describe your current beam line's capabilities.

Example:

```
smb1x5:~/release-3_2/blu-ice/data > cp periodic_bl92.dat periodic_bl92sim
```

3. Modify the `blu-ice/scripts/dice_tabs.tcl` file and change the following line to reflect a connection to your own Diffraction Image Server set up in Section 4.6.

```
blctl92:~/release-3_2/blu-ice/scripts > cvs diff
cvs diff: Diffing .
Index: dice_tabs.tcl
=====
RCS file: /home/code/repository/blu-ice/scripts/dice_tabs.tcl,v
retrieving revision 1.39
diff -r1.39 dice_tabs.tcl
136c136
< Diffimage lastImage $gWindows(collect,diffimage,frame) 134.79.31.20 14001
---
> Diffimage lastImage $gWindows(collect,diffimage,frame) localhost 14001 50
```

The 14001 in the above example should correspond to *guiPort* which is specified on the command line when starting the Diffraction Image Server.

4. Configure your `TCLLIBPATH` environment variable to grab the needed TCL libraries. For example:

```
setenv TCLLIBPATH "/home/scottm/release-3_2/widgets ; /usr/local/BWidget"
```

5. Modify the first line of `blu-ice/scripts/genkey.sh` to point at the location of your platform's `tcsh` binary.

```
smb1x5:~/release-3_2/blu-ice/scripts > which tcsh
/bin/tcsh
```

6. Create a directory `home/adm/keys` which is world writable, on a file system which DCSS can access as well.
7. Start BLU-ICE with the name of the beam line that you wish to connect to.

```
~/release-3_2/blu-ice/scripts/ice.tcl beamlineName
```

where the *beamlineName* will be used in the `switch` statement within the `ice.tcl` file to configure the Blu-Ice.

4.9 Installing SSRL's DHS

A DHS can be any program that can speak the DCS protocol (or subset of it) and does one of the following:

- Accepts DCS messages and controls a piece of hardware directly.
- Accepts, translates, and forwards DCS messages to another control system.

SSRL currently uses a DHS program which supports the following hardware:

- DMC2180 motion controller.
- ADSC Quantum 4 CCD detector.
- ADSC Quantum Q315 CCD detector.
- MAR345 detector without a base.
- Canberra DSA 2000 multi channel analyzer (limited functionality)
- Image analysis of JPEG's obtained by AXIS 2400 servers. (for automatic crystal centering)

This program requires that a functional mysql database is available so that it can obtain its configuration at start-up.

1. Install the mysql client library as described in Section 4.1.9.
2. Install the xos library as described in Section 4.1.1.
3. Install the diffimage library as described in Section 4.1.3.
4. Checkout dhs from CVS into your DCS 'root' directory.

```
blctlxx:~/release-3_2 > cvs checkout -r release-3_2 dhs
```

5. Check out the Q4 image transformation files found in the xform project. These files contain some algorithms and code which Area Detector Systems Corporation⁷ does not wish to have distributed without prior permission.

⁷<http://www.adsc-xray.com/>

```

blctlxx:~/release-3_2 > cvs checkout -r release-3_2 xform
cvs checkout: Updating xform
U xform/xform.c
U xform/xform.h
blctlxx:~/release-3_2 >

```

If you do not have access to this project, the `xformstub.c` and `xformstub.h` (provided by SSRL) can be used to build the DHS. Create the `xform` directory in your 'DCS root' directory before copying and renaming the follow the files as shown in this example:

```

blctlxx:~/release-3_2 > mkdir xform
blctlxx:~/release-3_2 > cd xform
blctlxx:~/release-3_2/xform > cp ~/release-3_2/dhs/src/xformstub.c xform.c
blctlxx:~/release-3_2/xform > cp ~/release-3_2/dhs/src/xformstub.h xform.h

```

6. Install the third party `dali` library or remove references to `-DWITH_CAMERA_SUPPORT` and `DALIIMAGE_LIBS` from the makefile if you do not need support for image analysis for crystal centering.

```

smbly5:~/release-3_2/dhs/linux > cvs diff
cvs diff: Diffing .
Index: dhs
=====
RCS file: /home/code/repository/dhs/linux/dhs,v
retrieving revision 1.1
diff -r1.1 dhs
Binary files /tmp/cvsB6eKRE and dhs differ
Index: makefile
=====
RCS file: /home/code/repository/dhs/linux/makefile,v
retrieving revision 1.11
diff -r1.11 makefile
31c31
< CPP_CONSTANTS = -DLINUX -Ddatabase_mysql -DWITH_CAMERA_SUPPORT -DWITH_DETE
---
> CPP_CONSTANTS = -DLINUX -Ddatabase_mysql -DWITH_DETECTOR_SUPPORT -DWITH_A
62,63c62
< ALL_LIBS = ${MYSQL_LIB_DIR}/libmysqlclient.a ${LOCAL_LIBS} ${THREAD_LIBS

```

```

<
---
> ALL_LIBS = ${MYSQL_LIB_DIR}/libmysqlclient.a ${LOCAL_LIBS} ${THREAD_LIBS} ${STD_L
81,83c80
<     safeFile.o \
<     dhs_Camera.o \
<     imgCentering.o
---
>     safeFile.o
smb1x5:~/release-3_2/dhs/linux >

```

7. Edit the `dhs_main.c` file and change the default mysql database host-name reference the hostname of your mysql server. For example, to change to localhost:

```

blctlxx:~/release-3_2/dhs/src > cvs diff
cvs diff: Diffing .
Index: dhs_main.cc
=====
RCS file: /home/code/repository/dhs/src/dhs_main.cc,v
retrieving revision 1.6
diff -r1.6 dhs_main.cc
70c70
<  m_db_hostname = "smb";
---
>  m_db_hostname = "localhost";

```

8. `cd` into the `linux`, `decunix`, or `irix` build directory and build.

```
blctlxx:~/release-3_2/dhs/linux > make
```

4.10 Configuring SSRL's DHS

At start-up the SSRL `dhs` attempts to connect to the mysql database to determine what its configuration is (i.e. what it is controlling).

The following subsections describe the steps for configuring a simple beam line.

1. ADD permissions to the mysql database for the new computer for which the new `dhs` is running on.

```
mysql> grant select on * to dhs@blctlxx.slac.stanford.edu;
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.42 sec)
```

2. Find out which beam line the DHS is going to be a part of. Use this number in the following step when entering the information for the BeamlineID.

```
mysql> select * from DCSS where BeamlineID=12;
+----+-----+-----+-----+-----+
| id | HostName                | ListeningPort | MemoryMapFile | BeamlineID |
+----+-----+-----+-----+-----+
| 12 | blctlxx.slac.stanford.edu |          14242 | database.dat   |           1 |
+----+-----+-----+-----+-----+
1 row in set (0.02 sec)
```

3. Insert an entry in the mysql DHS table and use the id in the next step. The fields in this table are used for the following purposes:

- **HostName:** Name of the host computer that DHS will run on
- **privateNode:** This is the hostname for which a network based piece of hardware should reconnect to if it needs to make an independent socket connection back to the DHS for some reason. The Galil DHS uses this to send asynchronous messages back to the DHS when the shutter opens during an oscillation. For many computers this field could be the same as the HostName, but on SSRL machines (which often have two network interfaces), it is important to be specific.
- **BeamlineID:** Should be the same as the BeamlineID entry in the DCSS table.
- **LocalDatabaseFileName:** This is the name of a file which will be created automatically by the DHS on start-up if it does not exist. It is a memory mapped file which will store motor positions if controlling a motion controller. For DHS's that only control detectors or other devices without persistent state, the file still gets generated but doesn't really do anything.

- **AutoFlushPeriod:** Specifies how often the `LocalDatabaseFileName` should be updated to disk in ms.
- **DevicePollPeriod:** Specifies in ms how often to poll and update to DCSS a moving motor.
- **WatchdogKickPeriod:** The galil is running a script which will stop all motors if it does not receive a 'kick' from the DHS every 5(?) seconds. This value tells the DHS how often to 'kick' the watchdog.
- **InstanceName:** This should be the name of the controlled device which will be specified at the command prompt when starting dhs. For example, the dhs program could be starting like this:
./dhs detector.

```
mysql> select * from DHS where BeamlineID=12;
```

```
+-----+-----+-----+-----+-----+
| id | HostName                | privateNode | BeamlineID | LocalDatabaseFileName |
+-----+-----+-----+-----+-----+
| 32 | blctlxx.slac.stanford.edu |             | 12         | blxxDetector.dat      |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

4. Add as many entries in the `Controllers` table as the number of new devices that this DHS is responsible for. The `id` field of the `Controllers` table will be used in entries in the `ControllerID` field of the device specific tables.

If the DHS is responsible for only one device the table may look like this:

```
mysql> select * from Controllers where DHS_ID=32;
```

```
+-----+-----+
| id | DHS_ID |
+-----+-----+
| 35 | 32     |
+-----+-----+
1 row in set (0.00 sec)
```

If the DHS is responsible for more than one devices, the table may look like this:

```
mysql> select * from Controllers where DHS_ID = 32;
+----+-----+
| id | DHS_ID |
+----+-----+
| 35 |      32 |
| 36 |      32 |
| 37 |      32 |
+----+-----+
24 rows in set (0.00 sec)
```

5. Find the table representing the new hardware component. For example, if you are installing a new CCD (or emulated CCD), you would want to add an entry to the CCD table. Add all of the relevant facts about the hardware component in this table.

```
mysql> select * from CCD where ControllerID=35;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | serialNumber | HostName                | CommandPort | DataPort | Ch  |
+----+-----+-----+-----+-----+-----+-----+-----+
| 9  | 0            | blcpu3.slac.stanford.edu |      8041   |      8042 |    |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

4.10.1 Miscellaneous show-stoppers for DHS

1. DCSS must be explicitly configured to expect the DHS' connection.

This is done by changing DCSS' `database.dat` file to expect the connection from the new DHS by making a DHS entry as discussed in Section 5.3.3.

4.10.2 Starting and Stopping DHS

Start the DHS from the build directory.

```
blctlxx:~/release-3_2/dhs/linux > ./dhs instanceName
where instanceName corresponds to the InstanceName in the DHS table
in the mysql database.
```

There are no specific procedure for stopping DHS. It is recommended, of course, to wait for motors to stop moving before hitting control-c or sending the kill signal.

4.11 Installing the detector simulator

The Q4 CCD detector portion of the DHS can be used with a `det_emulator` program which can be obtained separately from Area Detector Systems Corporation⁸. The `det_emulator` program simulates a Q4 (or actually a Q1) detector, allowing a simulated beam line to collect 'data'.

The `det_emulator` generates artificial images and sends the images over socket connections following the same protocol as a the `det_api` program. The detector DHS is responsible for writing the images to disk.

1. First install and configure a DHS as described in Section 4.9.
2. Copy the correction files from somewhere for the simulated CCD (step not shown).

```
blctlxx:~/release-3_2/dhs/linux > ls -alrt POSTNUF CALFIL NONUNF
-rwxr-----  1 blctl  postgres  334520 May 23 10:53 POSTNUF
-rwxr-----  1 blctl  postgres 25812118 May 23 10:53 CALFIL
-rwxr-----  1 blctl  postgres 10617344 May 23 10:53 NONUNF
```

3. Copy the CCD emulator and environment file.

```
smbdcsdev:/usr/local/dcs/det_emulator >scp blcpu2:~/release-3_2/dhs/det_emulator .
det_emulator          100% |*****| 81920      00:00

smbdcsdev:~/release-3_2/det_emulator >scp blcpu2:~/release-3_2/dhs/CCD_API.ENV .
CCD_API.ENV           100% |*****| 198        00:00
smbdcsdev:~/release-3_2/det_emulator >
```

4. Start the CCD emulator

```
smbdcsdev:~/release-3_2/det_emulator >det_emulator
-----
                CCD Detector Control V1.1
                Copyright ADSC 1997
-----

local column offset set to 35 for detector 0
local column offset set to 69 for detector 1
```

⁸<http://www.adsc-xray.com/>

```
local column offset set to 20 for detector 2
local column offset set to 17 for detector 3
ccdnet_init: CCD size is: 1242 (rows) x 1152 (cols)
ccdnet_init   : at Dec 19 15:03:51: initialized system for controller: 0
ccdnet_init   : at Dec 19 15:03:51: system properly initialized.
```

5. Start up the detector dhs.

```
blctlxx:~/release-3_2/dhs/linux > ./dhs detector
```

5 Maintaining the database.dat file

A properly configured DCSS requires a `database.dat` file.

The `database.dat` file contains the following information:

- List of all DHS programs allowed to connect to DCSS.
- List of all devices controlled by DCSS.
- Each device's last known position and/or state.
- Name of the DHS controlling each device.
- All other dynamic information regarding the device (e.g. software limits, scale factors, etc)
- Run definitions.

The `database.dat` file is memory mapped, allowing the operating system to keep the file as up-to-date as possible. Because the file is memory mapped, it is not possible to edit the file directly with a text editor. Instead, DCSS has the ability to transform the `database.dat` file into a text file. The text file can be edited and DCSS can create a new `database.dat` file from the modified file.

5.1 Dumping the database.dat file

To convert the contents of the `database.dat` file to text:

1. Stop DCSS.
2. `cd ~/release-3_2/dcss/linux`

3. `./dcss -d fileName`
4. The file *filename* should now contain the contents of the `database.dat` in text format.

5.2 Recreating the `database.dat` file

Performing this function is an easy way to *LOSE ALL OF YOUR MOTOR POSITIONS!*

To convert the contents of properly formatted text file into a `database.dat` file:

1. Stop DCSS and/or make sure that no DCSS is running.
2. `ps -ef | grep dcss`
Verify that DCSS is not running.
3. `cd ~/release-3_2/dcss/linux`
4. `./dcss -r fileName`
where *fileName* is the name of a properly formatted configuration file.
5. The `database.dat` file has now been created with the new configuration.

5.3 Format of the `database.dat` file

The text file accepted by the `dcss -r` command is a list of entries. Each entry is separated by a blank line (carriage return).

Each entry consists 3 or 4 lines of data describing the entry.

1. The first line of an entry is the *device name* as known by DCSS.
The *device name* may consist of regular alpha/numerical characters and the underscore character.
2. The second line is an integer representing the *device type* as shown in Table 1.
3. The contents of the remaining lines depend upon the entry for the *device type* as discussed in the following subsections.

Device type	description
1	Real Motor
2	Combo Motor
3	DHS description
4	Ion chamber
5	<i>Obsolete</i>
6	Shutter/Filter
7	<i>Obsolete</i>
8	Run Definition
9	Runs Definition
10	<i>Obsolete</i>
11	Operation
12	Encoder
13	String

Table 1: Device Types in database.dat

5.3.1 The Real Motor Entry

A real motor is a motor controlled by a hardware DHS.

An entry for a Real Motor definition should be in the following generic format:

```

Line 1: motorName
Line 2: 1
Line 3: responsibleDHS externalMotorName
Line 4: position upperLimit lowerLimit scaleFactor speed acceleration
backlash lowerLimitOn upperLimitOn motorLockOn backlashOn reverseOn
circleMode
Line 5: 0
Line 6: passiveOk remoteOk localOk inHutchOk closedHutchOk
Line 7: passiveOk remoteOk localOk inHutchOk closedHutchOk

```

- The *responsibleDHS* must be the name of a DHS defined in separate entry as discussed in Section 5.3.3.
- The *externalMotorName* parameter can be the same name as the *motorName*, but is provided here for mapping the motor name to a DHS(s) that names the motor differently. This is common with the motors controlled by the ICS system.
- The *position* parameter is the current position of the motor.

- The *upperLimit* and *lowerLimit* parameters are the software limits for the motor. The limits are only used if the corresponding *upperLimitOn* and *lowerLimitOn* flags are enabled.
- The *scaleFactor* converts motor steps to the default units of the motor. Note that the units of the motor are not listed in this file.
- The *speed* parameter is in units of steps/second.
- The *acceleration* parameter is in units of milli-seconds.
- The *backlash* parameter is in steps.
- The *lowerLimitOn* parameter can be 0 (for no software limit) or 1 (to enable the *lowerLimit* parameter).
- The *upperLimitOn* parameter can be 0 (for no software limit) or 1 (to enable the *upperLimit* parameter).
- The *motorLockOn* parameter can be 0 (which allows the motor to move) or 1 (prevents the motor from moving.)
- The *backlashOn* parameter can be 0 (no backlash) or 1 (uses the *backlash* parameter for each move.)
- The *reverseOn* parameter can be 0 (no internal swapping of direction) or 1 (internally swaps the direction of the motor move.)
- The *circleMode* parameter can be 0 (linear motion) or 1 (automatically converts 360 degrees to 0, and automatically picks the shortest path of motion.)
- The two permissions lines (lines 6 and 7) are discussed in Section 5.3.11.

Here is an example for `table_vert_1` motor entry. It is is motor controlled by a DHS called `gi`. The DHS `gi` knows this motor by a different name `tablev1`.

```
table_vert_1
1
gi tablev1
23.099118 49.999984 0.000000 3145.921000 500 125 1573 0 0 0 1 0 0
0
0 1 1 1 1
0 1 1 1 1
```

5.3.2 The Pseudo Motor Entry

A pseudo motor is device that can be moved like a regular motor, but does not necessarily control a single real motor. The pseudo motor may actually move several motors in combination, or perform additional functions and checks while moving the motor. A pseudo motor controlled by the scripting engine is called a scripted device.

An entry for a Pseudo Motor definition should be in the following generic format:

```
Line 1:motorName  
Line 2:2  
Line 3:responsibleDHS externalMotorName  
Line 4:position upperLimit lowerLimit lowerLimitOn upperLimitOn  
motorLockOn circleMode  
Line 5:0  
Line 6:0  
Line 7:passiveOk remoteOk localOk inHutchOk closedHutchOk  
Line 8:passiveOk remoteOk localOk inHutchOk closedHutchOk
```

- The *responsibleDHS* must be the name of a DHS defined in separate entry as discussed in Section 5.3.3.
- The *externalMotorName* parameter can be the same name as the *motorName*, but is provided here for mapping the motor name to a DHS(s) that names the motor differently. This is common with the motors controlled by the ICS system.

This parameter can also be used by the scripting engine to locate a specialized script for the motor. For example, using `standardVirtualMotor` would indicate to the `self` dhs that it should use the specialized `standardVirtualMotor` template when working with this motor.

- The *position* parameter is the current position of the motor.
- The *upperLimit* and *lowerLimit* parameters are the software limits for the motor. The limits are only used if the corresponding *upperLimitOn* and *lowerLimitOn* flags are enabled.
- The *lowerLimitOn* parameter can be 0 (for no software limit) or 1 (to enable the *lowerLimit* parameter).
- The *upperLimitOn* parameter can be 0 (for no software limit) or 1 (to enable the *upperLimit* parameter).

- The *motorLockOn* parameter can be 0 (which allows the motor to move) or 1 (prevents the motor from moving.)
- The *circleMode* should be always be 0. See bug 39⁹.
- The two permissions lines (lines 7 and 8) are discussed in Section 5.3.11.

5.3.3 The DHS definition

An entry for a complete DHS definition should be in the following generic format:

```
Line 1:name
Line 2:3
Line 3:hostName protocolLevel
```

- The *hostName* parameter is the host name of the computer that DHS will be running on. DCSS will reject a connection from a DHS announcing itself with the *name* if the connection is not coming from the computer/hostname *hostName*.
- The *protocolLevel* can be a 1 or a 2. A protocol of 1 is used by DHS(s) that are still using the 200 byte DCS protocol. A protocol of 2 is used by DHS(S) that have been upgraded to the dynamic length DCS protocol.

Here is an example for a **camera** DHS entry:

```
camera
3
biotest.slac.stanford.edu 2
```

Note: Every `database.dat` file should have an entry for the scripting engine DHS as shown here:

```
self
3
localhost 2
```

This will define a **self** DHS which `scripted operations` and `scripted devices` should use.

⁹https://smb.slac.stanford.edu/bugzilla/long_list.cgi?buglist=39

5.3.4 The Ion Chamber Entry

This entry maps very closely to the ICS control system's way of defining a ion chamber.

```
Line 1:ionchamberName  
Line 2:4  
Line 3:responsibleDHS counter counterChannel timer timerType
```

- The *responsibleDHS* must be the name of a DHS defined in separate entry as discussed in Section 5.3.3.
- The *counter* is the name of a counter in ICS.
- The *counterChannel* is the counter's channel in ICS.
- The *timer* is ?
- The *timerType* is ?

Here is an example ion chamber entry:

```
i_sample  
4  
simDhs hex1 3 rtc1 clock
```

5.3.5 The Shutter Entry

This entry can be used for devices that have binary state (e.g. shutters and filters)

```
Line 1:shutterName  
Line 2:6  
Line 3:responsibleDHS shutterState
```

- The *responsibleDHS* must be the name of a DHS defined in separate entry as discussed in Section 5.3.3.
- The *shutterState* is either a 0 (open) or a 1 (closed).

Here is an example shutter entry:

```
A1_4  
6  
galil2 0
```

5.3.6 The Run Definition Entry

There are 17 run definitions allocated in the `database.dat` file for the 17 possible runs that can be defined in the Collect tab in BLU-ICE.

An entry for a Run definition should be in the following generic format:

```
Line 1:runName  
Line 2:8  
Line 3:runStatus nextFrame runLabel fileroot directory startFrameLabel  
axisMotor startAngle endAngle delta wedgeSize exposureTime distance  
numEnergy energy1 energy2 energy3 energy4 energy5 modeIndex inverseOn
```

- The *runName* is usually `run0` through `run16`.
- The *runStatus* can be `collecting`, `paused` or `inactive`. If DCSS is stopped, a run with a status of `collecting` should be manually changed to `inactive` or `paused`.
- The *nextFrame* parameter is a positive integer that indicates how much of the run has been collected.
- The *runLabel* parameter is the text that appears on the tab. It also is used to generate the filename.
- The *fileroot* parameter is the text that is used as the root for the filename.
- The *directory* parameter is the directory that the user wishes to write the files into.
- The *startFrameLabel* parameter is a numerical value that is used as a starting point to generate a unique file name per frame.
- The *axisMotor* parameter is the motor name that will be used in the oscillation code. Usually this will be either `gonio_phi` or `gonio_omega`.
- The *startAngle* parameter is the starting angle of the *axisMotor* for the run definition.
- The *endAngle* parameter is the final ending angle for the *axisMotor*.
- The *delta* parameter is the distance that the *axisMotor* will travel per frame.

- The *wedgeSize* parameter is the angular distance that the *axisMotor* must travel before rotating 180 degrees and collecting the frames in the inverse wedge.
- The *exposureTime* parameter is the amount of time each frame will be exposed. This value is the requested time, and is not the dose corrected time.
- The *distance* parameter is the position for *detector_z* for each frame in the run.
- The *numEnergy* parameter is number of energy values that will be used during the collection of the run. This parameter should never exceed 5, as there are only 5 energy values that may be defined in a single run. A zero in this position would also be bad.
- The *energy1...energy5* parameters are the values for energy that will be used.
- The *modeIndex* parameter is the detector mode in an integer format. This parameter can have different meanings depending on the detector type being used. For example, detector mode 0 for a MAR345 is different than detector mode 0 for a CCD.
- The *inverseOn* parameter is a boolean value (0=FALSE or 1=TRUE) indicating whether or not the inverse wedge should be collected or not.

5.3.7 The Runs Definition entry

The Runs Definition entry is used to store information regarding data collection that will apply to all defined runs. There can only be one one Runs Definition in a *database.dat* file.

An entry for a Runs definition should be in the following generic format:

```
Line 1:runs
Line 2:9
Line 3:runCount currentRun isActive doseMode
```

- The *runCount* parameter indicates how many of the run definitions are valid. For example, a value of 4 would indicate that run0, run1, run2, and run3 device entries are valid. All other run definitions are not valid.

- The *currentRun* parameter is no longer used.
- The *isActive* parameter is no longer used.
- The *doseMode* is a Boolean value indicating whether or not to use a dose corrected exposure time per frame during data collection (0=no dose mode, 1=dose mode).

Here is an example runs entry indicating that two runs are defined and dose mode is disabled:

```
runs
9
2 2 0 0
```

5.3.8 The Operation Entry

An operation is a generic function. It does not have state stored in the database.dat file.

```
Line 1: operationName
Line 2: 11
Line 3: responsibleDHS externalName activeClientOnly
Line 4: passiveOk remoteOk localOk inHutchOk closedHutchOk
Line 5: passiveOk remoteOk localOk inHutchOk closedHutchOk
```

- The *responsibleDHS* must be the name of a DHS defined in separate entry as discussed in Section 5.3.3.
- The *externalName* parameter can be used by the scripting engine to locate a specialized script for the operation. In the case of the scripting engine, it appends a *.tcl* to this parameter to locate the script for the operation.
- The *activeClientOnly* parameter can be a 0 to allow any client to request the operation or 1 to allow only the active client to request the operation.
- The two permissions lines (lines 4 and 5) are discussed in Section 5.3.11.

Here is an example operation entry:

```
getLoopTip
11
camera getLoopTip 1
0 0 0 0 0
0 0 0 0 0
```

5.3.9 The Encoder Entry

An encoder entry allows DCSS to know which DHS has access to the encoder. It does not map an encoder to a real motor. The description of a motor's behavior in relationship to its encoder should be done using a scripted device.

```
Line 1:encoderName
Line 2:12
Line 3:responsibleDHS externalName
```

- The *responsibleDHS* must be the name of a DHS defined in separate entry as discussed in Section 5.3.3.
- The *externalName* parameter should probably be the same as the *encoderName*.

Here is an example of an encoder entry:

```
detector_z_encoder
12
simDhs detector_z_encoder
```

5.3.10 The String Entry

5.3.11 The Device Permissions Bits

The real motor, pseudo motor, and operation definition entries in the database.dat file each have two lines for restricting the use of the device. The two lines each have the same bits, but apply to different types of groups of users. The first line applies to the Blu-Ice user that have 'Staff' permissions. The second line applies to the Blu-Ice clients that do not have 'Staff' permissions.

There are two lines permission entry for each device will look like this:

```
Line x: passiveOk remoteOk localOk inHutchOk closedHutchOk
Line x+1:passiveOk remoteOk localOk inHutchOk closedHutchOk
```

1. The *passiveOk* parameter can be 0 or 1. A '0' indicates that the Blu-Ice client requesting the move must be 'Active' before DCSS forwards the request to the responsible DHS. A '1' in this field would allow DCSS to forward the motor move request to the responsible DHS even if the requesting Blu-Ice is not 'Active'.
2. The *remoteOk* parameter can be 0 or 1. A '0' indicates that with the hutch door open, the Blu-Ice client must be either running on a 'LOCAL' or 'HUTCH' console before DCSS will forward the move request to the responsible DHS. A '1' in this field would allow the move request to be forwarded to the DHS regardless of where the Blu-Ice client is running, even with the hutch door open.
3. The *localOk* parameter can be 0 or 1. A '0' indicates that with the hutch door open, the Blu-Ice client must be running on a 'HUTCH' console before DCSS will forward the move request to the responsible DHS. A '1' in this field would allow the move request to be forwarded to the DHS if the requesting Blu-Ice client is running on a 'LOCAL' machine, even with the hutch door open.
4. The *closedHutchOk* parameter can be 0 or 1. A '0' indicates that the hutch door must be open before the request to move the motor is forwarded to the responsible DHS. A '1' in this field indicates that the motor move may be moved with the hutch door closed.

Figure 1 shows the possible combinations of permission bits and the expected behaviour of the device depending on whether or not the client is 'REMOTE', 'LOCAL', or in the 'HUTCH', and whether or not the hutch door is closed or open.

6 Writing Scripted Devices and Operations

BLU-ICE and the scripting engine within DCSS are both written in the TCL scripting language. One of the features of a scripting language is that code does not need to be pre-compiled to be executed. This greatly simplifies the writing and testing of scripts. BLU-ICE itself allows the user to open a command prompt, which gives the user access to the TCL interpreter running BLU-ICE. New scripts can be loaded from this command prompt and executed, allowing complicated or repetitive tasks to be somewhat automated.

Device Restrictions					door closed		door open		
passiveOk	remoteOk	localOk	inHutchOk	closedHutchOk	REMOTE	LOCAL	REMOTE	LOCAL	HUTCH
x	0	0	0	0	n2	n2	n2	n2	n2
x	0	0	0	1	y	y	n3	n4	n5
x	0	0	1	0	n7	n7	n3	n4	y
x	0	0	1	1	y	y	n3	n4	y
x	0	1	0	0	n7	n7	n3	y	n5
x	0	1	0	1	y	y	n3	y	n5
x	0	1	1	0	n7	n7	n3	y	y
x	0	1	1	1	y	y	n3	y	y
x	1	0	0	0	n7	n7	y	n4	n5
x	1	0	0	1	y	y	y	n4	n5
x	1	0	1	0	n7	n7	y	n4	y
x	1	0	1	1	y	y	y	n4	y
x	1	1	0	0	n7	n7	y	y	n5
x	1	1	0	1	y	y	y	y	n5
x	1	1	1	0	n7	n7	y	y	y
x	1	1	1	1	y	y	y	y	y

- 1 NOT_ACTIVE_CLIENT
- 2 NO_PERMISSIONS
- 3 HUTCH_OPEN_REMOTE
- 4 HUTCH_OPEN_LOCAL
- 5 In_HUTCH_RESTRICTED
- 6 IN_HUTCH_AND_DOOR_CLOSED
- 7 HUTCH_DOOR_CLOSED

Invalid
Valid
Inconsistent
Consistent

Figure 1: Device Permissions

There are several drawbacks to running scripts through the command prompt of BLU-ICE:

- The BLU-ICE must remain open during the duration of the script.
- The BLU-ICE executing the script must remain the active client during the duration of the script.
- Other open BLU-ICE's will not know what the intentions of the script is, and therefore would not be able to interpret intermediate results.
- The BLU-ICE running the script must have full permissions to perform all actions within the script.
- The performance of the script depends on the local machine that is running the instance of BLU-ICE.

An alternative to running a script in BLU-ICE would be to run the script as an operation within the DCSS scripting engine. This feature overcomes all of the restrictions of running a script through BLU-ICE, but it does have several drawbacks of its own:

- The name of the script must be added manually to the `database.dat` file.
- Changes to the script requires DCSS to be shut down and restarted.
- A user's permissions must be checked within the scripting engine before each command is issued, because the scripting engine has unlimited privileges.

In many cases, development of new scripts can be done first through BLU-ICE and ported later to DCSS when the script has been tested sufficiently.

6.1 General DCS Scripting Commands

The scripting engine as well as the BLU-ICE command prompt offer the full command set of the TCL language. In addition to the standard TCL commands, BLU-ICE and the scripting engine add commands for controlling a beam line. These commands hide the details and complexity of the network protocol.

6.1.1 Writing to the log window in BLU-ICE

There are three commands for writing to the log window. All of the commands are written with a time stamp.

- `log_note comments`
Outputs a green comment to the log window.
- `log_warning warning`
Outputs a brown warning to the log window.
- `log_error error`
Outputs a red error message to the log window

6.1.2 Querying a motor position

Motor positions are available by adding a \$ in front of the name of the motor.

From the BLU-ICE command prompt, the user may type the following to obtain a motor position:

```
log_note $table_vert
```

Within a script it is necessary to include the following statement at the top of each procedure for each motor of interest: `variable motorName`.

Example: Writing the current motor position to the log window

```
proc print_some_motor_positions {} {  
variable table_vert  
variable gonio_phi  
  
log_note $table_vert  
log_note $gonio_phi  
}
```

Example Output:

```
15 Feb 2002 16:15:06 NOTE: 30.774772  
15 Feb 2002 16:15:16 NOTE: 21.000000
```

6.1.3 Moving a motor

The intuitively named `move` command is used to move a motor. The motor may be a scripted device or a real motor.

For scripts running within the scripting engine, the `units` field may be omitted and the scaled units will be used by default. The scripting engine does not currently¹⁰ know the units of the motor, and moves can only be made in the units described by the scale factor for the motor.

The general format of the command is:

```
move motorName by|to value units
```

The `units` parameter can be

- *scaled* for movement using the scale factor for the motor.
- *unscaled* for movement using motor steps
- *mm,um,A,eV,V,deg* if the command is being issued from BLU-ICE.

Examples:

Issuing move commands from BLU-ICE:

- `move gonio_phi to 30 deg`
Moves the motor `gonio_phi` to a position of 30 degrees.
- `move gonio_phi by 30 deg`
Moves the motor `gonio_phi` by 30 degrees from its current positions.
- `move gonio_phi to 30 unscaled`
Moves the motor `gonio_phi` by 30 steps.
- `move energy to 13000.00 eV`
- `move energy to 1.00 A`

Issuing move commands from the Scripting Engine:

- `move gonio_phi by 30 scaled`
Moves the motor to 30 degrees, if the scale factor for `gonio_phi` converts steps to degrees.
- `move energy to 13000.00`
The move will be in eV, if the script for `energy` reports its value in eV.

¹⁰https://smb.slac.stanford.edu/bugzilla/show_bug.cgi?id=23

6.1.4 Inserting/Removing shutters and filters

The DCS command for removing a filter is:

```
open_shutter shutterName
```

The DCS command for inserting a filter is:

```
close_shutter shutterName
```

Examples:

- `open_shutter shutter`
- `close_shutter Al_8`
- `open_shutter Se`

6.1.5 Waiting for hardware

An important aspect of writing scripts that interact with hardware is being able to guarantee that a hardware related function has completed before issuing the next command. The `wait_for_devices` command allows the script writer to halt the script at the appropriate time and wait for a hardware component to complete its current task. This command does not freeze the BLU-ICE GUI or the scripting engine while the script is waiting for the event that will trigger the `wait_for_devices` command to return.

The DCS command in its general form is:

```
wait_for_devices deviceName1 [deviceName2 [deviceName3 [...]]]
```

Each *deviceName* argument may be the name of a motor or ion chamber. The list of devices may be in any order.

An additional command is provided to wait for shutters/filters to be inserted and removed:

```
wait_for_shutters shutterName1 [shutterName2 [shutterName3 [...]]]
```

It should be understood that the `wait_for_shutters` command waits for the DHS responsible for the shutter to say that the shutter is open or closed. However, because there is often no shutter state feedback to a motion controller, the shutter may still be in the mechanical process of opening or closing.

At some point the functionality of `wait_for_shutters` should be added¹¹ to the `wait_for_devices` command, and the `wait_for_shutters` command should be phased out.

¹¹https://smb.slac.stanford.edu/bugzilla/show_bug.cgi?id=25

Examples:

- **Waiting for a motor to stop moving**

```
move gonio_phi by 30 deg
wait_for_devices gonio_phi
```

- **Waiting for multiple motors to stop moving**

```
#start two motors moving
move energy to 11000.00 eV
move table_vert to 22.3 mm

#wait for both motors to stop moving
wait_for_devices energy table_vert

#Both motors have stopped moving now.
```

- **Waiting for shutters/filters**

```
open_shutter shutter
open_shutter Al_1
wait_for_shutters shutter Al_1
```

6.1.6 Using ion chambers

The command for reading an ion chamber or chambers is

```
read_ion_chambers time detectorName1 [detectorName2 [...]]
```

All of the detectors must be associated with the same timer at the hardware level. After the command is issued, the `wait_for_devices` command can be issued to wait for the detectors to count over the requested time. After the `wait_for_devices` is issued, each of the readings for the ion chambers can be acquired individually using the `get_ion_chamber_counts` command in the following format:

```
get_ion_chamber_counts detectorName
```

Examples:

- **Reading two ion chambers for 2.5 seconds**

```
read_ion_chambers 2.5 i2 i5
wait_for_devices i2

#print the results to the log window

log_note [get_ion_chamber_counts i2]
log_note [get_ion_chamber_counts i5]
```

6.1.7 Using encoders

The command for reading an encoder position is:

```
get_encoder encoderName
```

The command for setting an encoder position is

```
set_encoder encoderName
```

After issuing the `get_encoder` command, it is necessary to issue a command to wait for the results to come back from the DHS responsible for this encoder. The command that does this is:

```
wait_for_encoder encoderName
```

This function should return the value of the encoder when the message is completed, but currently does not. This bug has a work around as described in the bug repository¹².

6.1.8 Using Operations

An operation can be a script executing by the scripting engine, or it can be hardware function executed by a hardware server. For example, reading out a detector may be a specific hardware function that can be accessed via an operation. The data collection script within DCSS is an example of a scripted operation.

6.1.9 Starting an operation

There are two forms of the command used to start operations:

```
start_operation          operationName [arg1 [arg2 [arg3 [...]]]]
start_waitable_operation operationName [arg1 [arg2 [arg3 [...]]]]
```

¹²https://smb.slac.stanford.edu/bugzilla/long_list.cgi?buglist=30

Both of these commands start the operation, but the `start_waitable_operation` will return a *operation handle*. This handle should be stored in a TCL variable and can be used later to obtain the results of the operation as discussed in section 6.1.10.

The number of needed arguments depends upon the operation that is being called.

Examples:

- `start_operation collectRuns 4`

This starts an operation `collectRuns` with a single argument of 4. There is no way for the caller to obtain results from the operation after initiating the operation in this way.

- `set opHandle [start_waitable_operation optimize table_vert i2 20 0.05 0.1]`

This starts an operation called `optimize` with several arguments. The TCL variable `opHandle` will contain a unique value which can be used later to obtain the results of the operation.

6.1.10 Obtaining operation results

The `start_waitable_operation` command returns a unique *operation handle*, which can be passed to the following command to obtain the results of the operation:

```
wait_for_operation operationHandle
```

This function will return the results of the operation in the following formatted string:

```
status returnValue1 [returnValue2 [returnValue3 [...]]]
```

The `status` field will contain one of the following:

- `normal`

This is returned if the operation has completed without any errors, and the return arguments are valid.

- `update`

This indicates that an update from the operation has been received. There may be arguments that can be used as intermediate results (e.g.

for graphing individual points as the operation proceeds). The `update` message may also be used as a trigger to perform another operation.

A status of `update` indicates that the operation is not yet completed. It is necessary to issued the `wait_for_operation` command again until the status changes to `normal`.

- An error message

Any string other than `normal` or `update` in the `status` field indicates that an error has occurred in the operation.

The `wait_for_operation` command will return a TCL level error which must be caught (using the TCL `catch` command) if the script writer intends to do any clean up. If the error is not caught, the currently running operation or scripted device will return an error condition.

See section 6.3 for details on handling errors.

It is not necessary to worry about the exact timing of the `wait_for_operation` command. If the `wait_for_operation` command is issued after the operation has already completed, the results of the operation will be returned by the scripting engine immediately. If the command is issued before the operation is completed, the script will hang until the operation is completed or the operation sends an `update` message.

Example: Waiting for the results of the optimize operation

```
set opHandle [start_waitable_operation optimize table_vert i2 20 0.05 0.1]
set result [wait_for_operation $opHandle]
log_note $result
```

output: normal 30.774772

6.2 Scripted Device Family Relationships

A scripted device must describe its relationship to other devices directly within its script. This is achieved using three commands: Device relationships are explicitly written within the scripts for the devices by using the following commands:

- `set_children`
- `set_siblings`

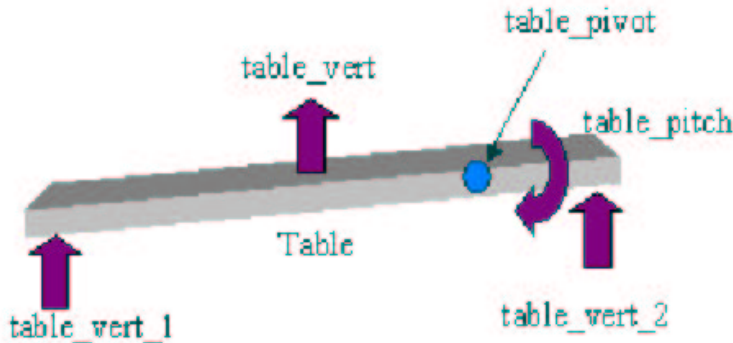


Figure 2: Children and Parent Motor Device

- `set_observers`

These commands should be issued from within the `deviceName_initialize` procedure as described in Section 7.1.

Figure 2 shows a typical example of several real motors being described by several pseudo motors. In this example, `table_vert` and `table_pitch` are parents of `table_vert_1` and `table_vert_2`. The parent motors `table_vert` and `table_pitch` are siblings (a.k.a joint custody parents).

6.2.1 Children and Parents

A scripted device is often used to control several real motors, or a hierarchy of scripted devices and real motors. If the relationship of the devices is such that a scripted device (`Device1`) should reevaluate its current position with each movement of (`Device2`), it is likely that `Device2` is a child of `Device1`, the parent device.

Within the script of the parent device, it is possible to declare this relationship with the following command:

```
set_children childDevice1 [[childDevice2] [...]]
```

After this command is issued, the scripting engine will automatically call the `deviceName_update` procedure each time a child motor moves. This procedure must be written by the script writer and should re-calculate and set the parent motor's new position. With the setting of the new position the scripting engine will then call all of the `deviceName_update` procedures for all scripted devices that are parents of this parent motor. Therefore, a single updated position from a child motor will cause all of the parents in the hierarchy to re-calculate their own new positions.

6.2.2 Siblings

Very often several scripted devices can be written to describe different relationships between the same real motors. If the user expects to be able to move each scripted device without affecting the position of the other scripted device, then the `set_siblings` command can be used to declare this desired relationship between two parent devices.

For example, the average position of `table_vert_1` and `table_vert_2` can be written into a scripted device called `table_vert`. Moving this `table_vert` “motor” would move `table_vert_1` and `table_vert_2` by an equal amount. However, `table_vert_1` and `table_vert_2` can also be described in terms of an angle, `table_pitch`. With the `table_vert` and `table_pitch` motors now defined, a user may expect to be able to move `table_vert` without changing `table_pitch`. On the other hand, a user may also expect to be able to move `table_pitch` without changing `table_vert`.

A problem arises with this scenario when one considers that the `table_vert` motor issues two independent move commands to two real motors. These two motors may travel at slightly different speeds, thus altering the `table_pitch` motor during the move. At the end of the move, the `table_pitch` should once again be back to its original position before the move was started. This by itself is probably acceptable, as the limits of `table_pitch` are continuously checked and the move would be aborted if the limits were exceeded.

However, if a move of `table_vert` is aborted, either by a user or by a scripted device that has had its limits violated, the position of `table_pitch` will have been altered at the end of the move. Further moves of `table_vert` would maintain the new altered position of `table_pitch`. This is unacceptable because the `table_pitch` motor was altered simply because of an interrupted move of `table_vert`.

The `set_siblings` command can help the scripting engine overcome this problem by doing the following:

- With each move a motor that completes normally, the current position of all of the motor’s siblings are stored as the last good position.
- With each move a motor that completes with an error (aborted or limit violation), no value will be stored.
- Every time a motor move is started, the status of all of the sibling motors are checked, and any of these motors that are not at the last good position will be told to move to their last good position before the initial request move is performed.

With `table_pitch` defined as a sibling of `table_vert`, aborting `table_vert` during a move will not update `table_pitch`'s last good position. When `table_vert` is moved again, the `table_pitch` motor will first be moved to its last good position before `table_vert` is allowed to move.

6.2.3 Observers

6.3 Exception Handling

As most developers of complex systems know, it is often a huge challenge to build in the ability to handle unexpected events. The designer should always be asking what the software should do when events like the following occur:

- A motor hits a hardware limit during a complex operation.
- The user hits the `abort` button during a complex operation.
- The computer running a DHS is powered off accidentally.
- The user issues a command with strange parameters that triggers a division by zero.
- A motion controller's power supply blows out.
- A vital network switch is re-booted.

The scripting engine within DCSS attempts to handle unexpected errors in a consistent way without requiring the script writer adding a single line of code. The writer of the script only needs to write the script as if everything is working normally to obtain the default error handling behavior. However, the default handling of exceptions may be insufficient in some situations, and the script writer can write additional code to handle these special cases.

All un-handled exceptions within a script will do one of two things, depending on the script type:

- For scripted operations, the script will terminate and will automatically append the TCL error to the operation's terminating DCS message.

This will allow the caller to be able to determine the reasons for scripted operation's failure.

- For scripted devices, the script will terminate and will automatically return a `aborted` status to the caller.

This is a system deficiency that will be overcome in later versions of DCS. The error message should be generic in the same way as the scripted operations.

Currently the scripting engine design is very picky about errors on motors and may seem heavy-handed. However, the purpose is to prevent possible hardware damage instead of allowing a script to grind onward by repeating a move again and again. The script writer should be careful about added exceptions handlers for the code that could override a default system behavior that was designed with safety in mind. Often it is best to allow the script to exit and force the user to resolve the problem.

6.3.1 The legalities of problems with motors

When issuing a `move` command from a script in the scripting engine, the following situations will cause an exception to be thrown:

- The global abort was issued after this script was started.
- The motor is already moving.
- The move would exceed the motor's software limits.
- The motor has a child motor that is currently moving.
- The motor is locked.

The following errors will cause the scripting engine to issue a global abort.

- A sibling motor needs to be moved back to its last good position, but the `move` command threw an exception for one of the reasons listed above.
- The scripting engine receives a message indicating that a motor has completed its move abnormally,

After successfully issuing a `move` command, the script will continue until the `wait_for_devices` function is called. The `wait_for_devices` command will return and allow the script to continue only if the device completed its move normally. If the device has not completed normally, an exception will be thrown with the reason that it failed.

6.3.2 Handling the global abort

Currently, the `Abort` feature within DCS is a global function, and abort messages will be sent to all active operations and moving motors. It is not possible to issue an abort to a particular motor. Future developments may make the aborting of an individual motor or operation a requirement, but this is not the current design philosophy.

After the scripting engine receives an `abort` message it will not allow a currently running script to start a motor moving or start another operation. The `move`, `start_operation`, and `start_waitable_operation` commands will throw an exception if they are called after the system has been aborted. The script is allowed to continue executing up to the point where it tries to issue one of these commands.

6.3.3 Handling DHS Crashes

The scripting engine will automatically generate terminating DCS messages for operations that are active and motors that are moving if the responsible DHS loses its socket connection. The terminated DCS message for each moving motor and active operation contains an error code, and an exception will be thrown for all of the outstanding `wait_for_devices` and `wait_for_operations` calls associated with the crashed DHS.

6.3.4 Throwing your own exceptions

The scripting engine will catch all un-handled errors and append these errors to the script's terminating DCS message. Therefore, the script writer may also throw exceptions that will automatically be transmitted out on the operation's completed message. For example, the following code within a script would terminate the script with a completed message and an error of `negativeNumber`. BLU-ICE clients could interpret this message and handle it however they wished.

```
if { $value < 0 } {  
  return -code error negativeNumber  
}
```

6.3.5 Writing specialized exception handlers

All exceptions generated within the scripting engine obey the rules of TCL. Therefore, the script writer should consult the TCL documentation to fully understand the syntax of exception handling.

If the script writer wishes to start an operation after a global abort has been issued, there are two commands for doing this:

- `start_recovery_operation operationName [arg1 [arg2 [arg3 [...]]]]`
- `start_waitable_recovery_operation operationName [arg1 [arg2 [arg3 [...]]]]`

These two functions behave the same as the `start_operation` and the `start_waitable_operation` commands, except that they bypass the check for the global abort flag.

These functions can be used to start operations that are needed in order to recover from a bad state.

For example, the `collectRun` operation interfaces with a Q4 CCD detector. The Q4 CCD detector should not flush its buffer after every image, because this would eliminate the double buffer capability of the detector. However, if an exception happens within the script somewhere, the detector must flush its buffer before the script is completed. Therefore the `collectRun` operation catches all exceptions, and starts the recovery operation `detector_stop` before throwing the original exception again.

7 Adding Scripts to the Scripting Engine

Adding new scripts to the DCSS scripting engine involves several steps as outlined in the following sub-sections.

7.1 Adding New Scripted Devices

This section describes the procedure for creating a new scripted device.

1. Create an entry in the `database.dat` file for the new device as described in Section 5.3.2.
 - Set the *responsibleDHS* to 'self'.
 - Set the *externalName* parameter to the name of your new device file without the `.tcl` extension.

Alternatively you can set the *externalName* to `standardVirtualDevice`. This would make the device a motor that simply holds a position. If you select the `standardVirtualDevice`, then this is the only step that you need to perform, and you can restart DCSS.

2. Create the new device file in the `dcss` devices directory: `dcss/scripts/devices`

The file name should be the name of the new device (as listed in the `database.dat` file) with a `.tcl` extension. For example, the `table_vert` scripted device should have a file `table_vert.tcl` within the devices directory.

3. Within the new scripted device file define 5 TCL procedures, where `deviceName` is replaced by the name of the new scripted device.

- `proc deviceName_initialize {}`

This procedure is executed when DCSS starts up. This procedure may be empty, or it may be used to initialize variables associated with the scripted device.

This is also the correct place to call the `set_children`, `set_siblings`, and `set_observers` functions as discussed in Section 6.2.

- `proc deviceName_set { newPosition }`

This is the procedure that is called when a motor configuration is initiated by the `set` command. The value being applied to the scripted device motor is passed in the `newPosition`. This procedure does not necessarily have to accept the new position, but can use the value to set other motors.

- `proc deviceName_move { newPosition }`

This is the procedure that is executed when an attempt is made to move the scripted device. The script is free to do whatever it wants, including moving other motors or calling scripted operations. If the script moves other motors, it is likely that these motors should be listed as children using the `set_children` command. However, this is not a requirement to moving other motors.

- `deviceName_update`

This procedure is called whenever the scripting engine has reason to believe that the scripted device's current position is out of date. This happens under the following conditions:

- The scripting engine receives an update on a child motor of the scripted device.
- The scripting engine receives an move complete on a child motor of the scripted device.

- The scripting engine receives a configuration on a child motor of the scripted device.

- `proc deviceName _calculate {[child1 [child2 [child3]]]}`

This procedure is called by the scripting engine when determining if a child motor is allowed to move to a certain position without violating the parent’s software limits. The arguments of this function are listed in the order that this function listed its children in the `set_children` command.

This function must accept theoretical children positions and recalculate a new position. It is common for the `deviceName _update` command to use this procedure to update its current position using current children motor positions.

4. Restart DCSS.

7.2 Adding New Scripted Operations

This section describes the procedure for creating a new scripted operation.

1. Create an entry in the `database.dat` file for the operation as described in Section 5.3.8.

Set the `responsibleDHS` to 'self'.

Set the `externalName` parameter to the name of your new operation file without the `.tcl` extension.

2. Create the new operation file in the `dcss` operation directory, `dcss/scripts/operations`
3. Within the new operation file define two procedures, where the `operationName` is replaced by the name of the new operation.

- `proc operationName _initialize {}`

This procedure is executed when DCSS starts up. This procedure may be empty, or it may be used to initialize variables associated with the scripted operation.

- `proc operationName _start { [arg1 [arg2 [arg3 [args]]]}`

The `operationName _start` procedure is executed when a `start_operation` message is received by the scripting engine. This procedure should contain the actual functionality of the operation.

The arguments passed to this procedure are the same arguments passed to the initiating command:

```
start_operation operationName [arg1 [arg2 [arg3 [...]]]].
```

4. Restart DCSS.

8 Example scripts

The following subsections should help give the reader ideas for writing their own scripted devices and operations.

8.1 Testing the diffractometer

It is possible to imagine a damaged diffractometer in which it is possible for the motorized sample motors to lose their encoder feedback lines in certain orientations of phi. The script in Figure 3 shows a diagnostics test that can be run from BLU-ICE to look for this type of situation. To run this script from BLU-ICE, open the command prompt and type

```
source filename
```

where the *filename* is the name of the file that contains the script.

This script will look for orientations of phi that disconnect the encoder lines. When the encoder lines are broken the sample motors would move until they hit a hardware limit, and the script will halt at the bad phi position.

If this script were found to be extremely useful and used often, the DCS administrator could convert this script into a scripted operation as shown in Figure 4. The operation would need to be added to the database.dat file as described in Section 7.2. The user of BLU-ICE would then be able to type in the following from the BLU-ICE command prompt to execute the operation:

```
start_operation diffractometerTest
```

8.2 The energy device on different beam lines

The differences between beam lines at SSRL can be easily addressed by writing specific scripts for each unique hardware architecture.

For example, beam line 9-2 and beam line 1-5 at SSRL each have their own energy script: `energy_b192.tcl` and `energy_b115.tcl` respectively. The `database.dat` file for each beamline has an `energy` device entry which points at the energy script for that beam line.

The entry for the `energy` scripted device may look like this in the `database.dat` file on beam line 9-2:

```

1 #store the current positions of the sample motors
2 set startx $sample_x
3 set starty $sample_y
4 set startz $sample_z
5
6 #initialize our phi variable
7 set phi 0
8 while {$phi < 360} {
9
10     log $phi
11
12     #test the next position of phi
13     move gonio_phi to $phi deg
14     wait_for_devices gonio_phi
15
16     #move the sample motors a little bit
17     move sample_x by 0.01 mm
18     move sample_y by 0.01 mm
19     move sample_z by 0.01 mm
20
21     #wait for the motors to finish the move
22     wait_for_devices sample_x sample_y sample_z
23
24     #move the sample motors back to start
25     move sample_x to $startx mm
26     move sample_y to $starty mm
27     move sample_z to $startz mm
28
29     #wait for the motors to finish moving back
30     wait_for_devices sample_x sample_y sample_z
31
32     #move our phi variable
33     set phi [expr $phi + 0.1]
34 }

```

Figure 3: Script for testing Diffractometer: Text Listing.

```

1  proc diffractometerTest_initialize {} {
2  }
3
4  proc diffractometerTest_start { } {
5      #Bring motor positions into this namespace
6      variable sample_x
7      variable sample_y
8      variable sample_z
9
10     #store the current positions of the sample motors
11     set startx $sample_x
12     set starty $sample_y
13     set startz $sample_z
14
15     #initialize our phi variable
16     set phi 0
17     while {$phi < 360} {
18
19         log $phi
20
21         #test the next position of phi
22         move gonio_phi to $phi
23         wait_for_devices gonio_phi
24
25         #move the sample motors a little bit
26         move sample_x by 0.01
27         move sample_y by 0.01
28         move sample_z by 0.01
29
30         #wait for the motors to finish the move
31         wait_for_devices sample_x sample_y sample_z
32
33         #move the sample motors back to start
34         move sample_x to $startx
35         move sample_y to $starty
36         move sample_z to $startz
37
38         #wait for the motors to finish moving back
39         wait_for_devices sample_x sample_y sample_z
40
41         #move our phi variable
42         set phi [expr $phi + 0.1]
43     }

```

Figure 4: Script for testing Diffractometer: Text Listing.

```

energy
2
self energy_bl92
14500.036128 15000.000000 5900.000000 0 0 0 0
0
0

```

whereas on beam line 1-5 the entry for `energy` may look like this:

```

energy
2
self energy_bl15
12999.846769 16000.000000 5900.000000 0 0 0 0
0
0

```

Beam line 9-2 moves the monochromator in order to change energy for the beam line. The monochromator is equipped with an encoder which allows the software to verify that the position is changing correctly. A `mono_theta_corr` scripted device (not shown here) was written which is responsible for moving the real `mono_theta` motor and verifies (using the encoder) that the motor achieved the desired position. The script for `energy` for beam line 9-2 shown in Figure 5 simply moves the `mono_theta_corr` device.

Beam line 1-5, on the other hand, requires moving a monochromator (without an encoder this time) as well as changing the voltage on a piezo. A scripted device was written to assist in controlling the voltage of the piezo according to a curve that was experimentally determined. In addition to this, the beam moves with the change in energy, and the table is adjusted slightly to accommodate this change. The resulting `energy` script is significantly different for beam line 1-5 as shown in Figure 6.

These scripts completely encapsulate the process of changing energy, hiding it as if the software were simply moving a motor. This greatly simplifies higher level software and allows the Blu-Ice Gui to provide a simple drop down menu for energy selection.

9 The DCS Protocol

Three components comprise the DCS architecture: hardware servers (DHS), GUI clients, and the DCS server, DCSS. GUI clients and hardware servers

```

1  # energy.tcl
2
3  proc energy_initialize {} {
4
5      # specify children devices
6      set_children mono_theta_corr d_spacing
7  }
8
9  proc energy_move { new_energy } {
10
11     # global variables
12     variable d_spacing
13
14     # calculate destination for mono_theta_corr
15     set new_mono_theta_corr \
16         [energy_calculate_mono_theta_corr $new_energy $d_spacing]
17
18     # move mono_theta to destination
19     move mono_theta_corr to $new_mono_theta_corr
20
21     # wait for the move to complete
22     wait_for_devices mono_theta_corr
23 }
24
25
26 proc energy_calculate { mtc ds } {
27
28     # calculate energy from d_spacing and mono_theta_corr
29     return [expr 12398.4244 / (2.0 * $ds * sin([rad $mtc]) ) ]
30 }
31
32
33 proc energy_calculate_mono_theta_corr { e ds } {
34
35     # calculate mono_theta from energy and d_spacing
36     return [deg [expr asin( 12398.4244 / ( 2.0 * $ds * $e ) ) ]]
37 }

```

Figure 5: Energy on Beam Line 9-2

```

1 # energy.tcl
2
3 proc energy_initialize {} {
4
5     # specify children devices
6     set_children mono_theta d_spacing mono_piezo table_pitch \
7         table_yaw table_vert table_horz
8 }
9
10 proc energy_move { new_energy } {
11
12     # global variables
13     variable energy
14     variable d_spacing
15     variable mono_piezo_offset
16     variable table_horz_1_offset
17     variable table_horz_2_offset
18     variable table_vert_1_offset
19     variable table_vert_2_offset
20
21     # calculate destination for mono_theta
22     set new_mono_theta [energy_calculate_mono_theta $new_energy $d_spacing]
23
24     # move mono_theta
25     move mono_theta to $new_mono_theta
26
27     # move mono_piezo
28     move mono_piezo to [expr \
29         [energy_calculate_mono_piezo $new_mono_theta] + $mono_piezo_offset]
30
31     # move table_horz_1
32     move table_horz_1 to [expr \
33         [energy_calculate_table_horz_1 $new_mono_theta] + $table_horz_1_offset]
34
35     # wait for the moves to complete
36     wait_for_devices mono_theta mono_piezo table_horz_1
37 }
38
39 proc energy_calculate { mt ds pv tp ty tv th } {
40
41     # calculate energy from d_spacing and mono_theta
42     return [expr 12398.4244 / (2.0 * $ds * sin([rad $mt]) ) ]
43 }
44
45 proc energy_calculate_mono_theta { e ds } {
46
47     # calculate mono_theta from energy and d_spacing
48     return [deg [expr asin( 12398.4244 / ( 2.0 * $ds * $e ) ) ]]
49 }
50
51 proc energy_calculate_mono_piezo { mt } {
52
53     expr 0.368 + 11.44397 * $mt - 0.208731 * $mt*$mt
54 }
55
56 proc energy_calculate_table_horz_1 { mt } {
57
58     expr -11.3647 + 0.0695967 * $mt - 0.0007411889 * $mt*$mt
59 }

```

communicate solely with DCSS, and DCSS operates the only listening sockets (i.e., network server) in the DCS system. Consequently, hardware servers are in a sense network clients of DCSS and are sometimes referred to as clients of DCSS in the DCS message protocol documentation.

DCS hardware servers encapsulate low-level control of physical devices such as motors, shutters, detectors, and so on. Hardware servers simply connect to DCSS and do whatever DCSS tells them to do. DCS GUI clients such as BLU-ICE are the ultimate source of these instructions to the hardware servers. DCSS passes the requests from GUI clients down to the appropriate hardware servers, and broadcasts all replies from hardware servers back to all of the GUI clients, thus keeping all GUI clients in complete synchronization.

By convention, the first four letters of the command in a DCS message specify the source and destination of the message in the DCS architecture. The five possibilities are the following:

- **stog**: server to gui client
- **stoh**: server to hardware client
- **gtos**: gui to server
- **htos**: hardware client to server
- **stoc**: server to client (hardware or GUI)

9.1 The DCS Message Structure

All communication between the components of the DCS architecture uses a message passing protocol which runs on top of TCP/IP. The protocol is completely asynchronous (i.e., not strict client/server).

The network message is split into three distinct sections: the header, the text section, and the binary section.

9.1.1 The DCS message header

Each DCS message must start with 26 bytes of text message. These 26 bytes must contain a 2 numbers in ASCII format. The two numbers indicate the size of the text section (in bytes) and binary section (in bytes). The text message must contain a terminating 0 at the end of the two numbers. This allows a library to simply use the `scanf` function to obtain the two numbers.

An example header may look like this:

9.2 Connection Protocol

The connection protocol has been simplified from the first two releases of DCS. The GUI and hardware clients connection to DCSS on different ports.

For hardware servers (DHS):

1. The new DHS opens a socket connection to the hardware listening port on DCSS.
2. DHS begins reading and handling messages sent from DCSS.
3. DCSS will send a `stoc_send_client_type` to the DHS.
4. The DHS has 1 second to respond with the following message:

`htos_client_is_hardware dhsName` as discussed in Section 9.4.1.

At this point DCSS will disconnect a DHS for the following reasons:

- The DHS does not respond within 1 second.
- The DHS does not respond with the name of a DHS listed within the `database.dat` file.
- There is already a DHS connected with the same name as the DHS that is currently trying to connect.

For GUI clients (BLU-ICE):

1. The new BLU-ICE opens a socket connection to the GUI client listening port on DCSS.
2. DCSS will send a `stoc_send_client_type` to the BLU-ICE.
3. The BLU-ICE has 1 second to respond with the following message:

`htos_client_is_gui userName hostname display`

Where *userName* is the unix account name of a the user that initiated the BLU-ICE.

At this point DCSS has the option to disconnect the BLU-ICE for the following reasons:

- The BLU-ICE does not send the response within 1 second.
- The BLU-ICE responds with a *userName* that is not listed within the mysql database `Users` table.

4. After DCSS confirms that the `userName` is listed in the mysql database, DCSS will send the following message to BLU-ICE with 200 bytes of trailing binary data:

`stog_respond_to_challenge`

5. BLU-ICE must read the 200 bytes of binary data and respond to DCSS appropriately.

At this point DCSS will disconnect the BLU-ICE for the following reasons:

- The BLU-ICE client does not respond within 1 second.
- The BLU-ICE does not send the appropriate response.

6. DCSS sends an message `stog_login_complete`

7. DCSS sends another message

`stog_set_permission_level permissionsLevel`

where *permissionsLevel* is a number obtained from the mysql's Users table that can be used by the GUI to disable or enable certain buttons and tabs. DCSS will also use this value to prevent the user from doing what they are not allowed to do.

8. DCSS will then proceed to update the BLU-ICE client with the complete state of the beam line. This will include every motor position.
9. DCSS will then begin processing message from BLU-ICE as they come in, and will send messages to the BLU-ICE client to keep the BLU-ICE up-to-date with the latest status of motor positions or operation activity.
10. DCSS will disconnect the BLU-ICE under the following circumstances:
 - The BLU-ICE is too slow in processing the message in its TCP/IP input buffer and DCSS's TCP/IP output buffer becomes full.
 - The BLU-ICE socket has an error.

9.3 Gui to Server Messages (gtos)

All commands sent from a GUI client to DCSS start with a `gtos` (pronounced g-2-s).

9.3.1 gtos_abort_all

The format of the message is

```
gtos_abort_all [hard / soft]
```

This command requests that all operations either cease immediately or halt as soon as possible. All motors are stopped and data collection begins shutting down and stopping detector activity. A single argument specifies how motors should be aborted. A value of *hard* indicates that motors should stop without decelerating. A value of *soft* indicates that motors should decelerate properly before stopping.

9.3.2 gtos_become_master

The format of the message is

```
gtos_become_master [force / noforce]
```

This command requests that the sender be made the **Active** GUI client. A mode of *force* indicates that the client wants to become the **Active** client even if another client is currently the **active** client. A mode of *noforce* indicates that it only wants to be **Active** if there currently is no other **Active** client.

DCSS responds with *stog_become_master* if the GUI succeeds in becoming the **Active** client. If *noforce* was specified by the GUI client and another GUI is currently **Active**, DCSS will reply with *stog_other_master*.

9.3.3 gtos_become_slave

The format of the message is

```
gtos_become_slave
```

This command requests that the sender be made a non-**Active** GUI client. DCSS responds with *stog_become_slave*.

9.3.4 gtos_configure_device

The format of the message can take one of two forms:

1. *gtos_configure_device motorName position upperLimit lowerLimit lowerLimitOn upperLimitOn motorLockOn*

- *motorName* is the name of the motor to configure.
- *position* is the scaled position of the motor.
- *upperLimit* is the upper limit for the motor in scaled units.

- *lowerLimit* is the lower limit for the motor in scaled units.
 - *lowerLimitOn* is a boolean (0 or 1) indicating if the lower limit is enabled.
 - *upperLimitOn* is a boolean (0 or 1) indicating if the upper limit is enabled.
 - *motorLockOn* is a boolean (0 or 1) indicating if the motor is software locked.
2. `gotos_configure_device motorName position upperLimit lowerLimit scaleFactor speed acceleration backlash lowerLimitOn upperLimitOn motorLockOn backlashOn reverseOn`

where

- *motor* is the name of the motor to configure
- *position* is the scaled position of the motor
- *upperLimit* is the upper limit for the motor in scaled units
- *lowerLimit* is the lower limit for the motor in scaled units
- *scaleFactor* is the scale factor relating scaled units to steps for the motor
- *speed* is the slew rate for the motor in steps/sec
- *acceleration* is the acceleration time for the motor in seconds
- *backlash* is the backlash amount for the motor in steps
- *lowerLimitOn* is a boolean (0 or 1) indicating if the lower limit is enabled
- *upperLimitOn* is a boolean (0 or 1) indicating if the upper limit is enabled
- *motorLockOn* is a boolean (0 or 1) indicating if the motor is software locked
- *backlashOn* is a boolean (0 or 1) indicating if backlash correction is enabled
- *reverseOn* is a boolean (0 or 1) indicating if the motor direction is reversed

This command requests that the configuration of a real motor be changed. DCSS updates the device configuration in its internal database (database.dat) and forwards the message to the appropriate hardware server.

Note: This message should probably be two separate messages `gotos_configure_real_motor` and `gotos_configure_pseudo_motor`.

9.3.5 gtos_read_ion_chambers

The format of the message is

```
gtos_read_ion_chambers time repeat ch1 [ch2 [ch3 [...]]]
```

where

- *time* is the time in seconds over which to integrate counts.
- *repeat* is a boolean (0 or 1) indicating if chamber should be read iteratively.
- *ch1..ch[n]* is the list of names of the ion chambers to read.

This command requests that one or more ion chambers be read, using the same real time clock. All of the specified ion chambers must be controlled by the same hardware server (DHS).

9.3.6 gtos_set_motor_position

The format of the message is

```
gtos_set_motor_position motorName position
```

where

- *motorName* is the name of the motor to configure.
- *position* is the new scaled position of the motor.

This message requests that the position of the specified motor be set to specified scaled value. It is similar to the `gtos_configure_device` but changes the position only. DCSS forwards this message to the appropriate hardware server(DHS).

9.3.7 gtos_set_shutter_state

The format of the message is

```
gtos_set_shutter_state shutterName state
```

where

- *shutterName* is the name of the shutter/filter to open or close.
- *state* is open or closed.

This message requests that the state of the specified shutter or filter be changed to a particular state.

9.3.8 gtos_start_motor_move

The format of the message is

```
gtos_start_motor_move motorName destination
```

where

- *motorName* is the name of the motor to move
- *destination* is the scaled destination of the motor.

The message requests that the specified motor be moved to the specified scaled position.

9.3.9 gtos_start_oscillation

The format of the message is

```
gtos_start_oscillation motorName shutter deltaMotor deltaTime
```

where

- *motorName* is the name of the motor to oscillate during the exposure.
- *shutter* is the name of the shutter used to expose the sample.
- *deltaMotor* is the range of motion that the exposure should occur.
- *deltaTime* is the time over which the exposure should occur.

The message requests that a precision exposure be performed using the specified motor and shutter, and over the motor range and time interval specified. The speed of the motor over the deltaMotor range should be uniform.

DCSS forwards this message to the appropriate hardware server as a `stoh_start_oscillation` message.

Note: This message is primarily used for testing oscillation code in hardware servers from BLU-ICE. It is not currently used to perform data collection sequencing from the GUI client level.

Note: This message will be converted to an operation message and will be obsolete in later versions of DCS.

9.3.10 gtos_start_operation

The format of the message is

```
gtos_start_operation operationName operationHandle [arg1 [arg2
[arg3 [...]]]]
```

where

- *operationName* is the name of the operation to be started.
- *operationHandle* is a unique handle currently constructed by calling the `create_operation_handle` procedure in BLU-ICE. This currently creates a handle in the following format:

```
clientNumber.operationCounter
```

where `clientNumber` is the number provided to the BLU-ICE by DCSS via the `stog_login_complete` message. DCSS will reject an operation message if the `clientNumber` does not match the client. The `operationCounter` is a number that the client should increment with each new operation that is started.

- *arg1* [*arg2* [*arg3* [...]]] is the list of arguments that should be passed to the operation. It is recommended that the list of arguments continue to follow the general format of the DCS message structure (space separated tokens). However, this requirement can only be enforced by the writer of the operation handlers.

The message requests DCSS to forward the request to the appropriate hardware server.

9.4 Hardware to Server Messages (htos)

All commands sent from a hardware server (DHS) to DCSS start with a `htos` (pronounced h-2-s).

9.4.1 htos_client_is_hardware

This should be sent by all hardware servers in response to `stoc_send_client_type` message from DCSS.

The format of the message is

```
htos_client_is_hardware dhsName
```

where

Where *dhsName* is the name of a hardware server listed within the `database.dat` file as described in Section 5.3.3.

Note: This message is not forwarded to the GUI clients.

9.4.2 htos_configure_device

This is a message from a hardware server to DCSS updating the configuration of a device. DCSS updates its internal database and forwards the message to all GUI clients. Arguments are the same as for `gtos_configure_device` as discussed in Section 9.3.4.

9.4.3 htos_motor_move_completed

Indicates that the move on the specified motor is now complete. DCSS forwards this message to all GUI clients as a `stog_motor_move_completed` message.

The format of the message is

`htos_motor_move_completed` *motorName position completionStatus*

- *motorName* is the name of the motor that finished the move.
- *position* is the final position of the motor.
- *completionStatus* is the status of the motor with values as shown below:
 - `normal` indicates that the motor finished its commanded move successfully.
 - `aborted` indicates that the motor move was aborted.
 - `moving` indicates that the motor was already moving.
 - `cw_hw_limit` indicates that the motor hit the clockwise hardware limit.
 - `ccw_hw_limit` indicates that the motor hit the counter-clockwise hardware limit.
 - `both_hw_limits` indicates that the motor cable may be disconnected.
 - `unknown` indicates that the motor completed abnormally, but the DHS software or the hardware controller does not know why.

9.4.4 `htos_motor_move_started`

This message indicates that the requested move of a motor has begun. This message is forwarded by DCSS to all GUI clients as a `stog_motor_move_started` message.

The format of the message is

```
htos_motor_move_started motorName position
```

where

- *motorName* is the name of the motor.
- *position* is the destination move the motor.

9.4.5 `htos_update_motor_position`

This message updates the position of a motor during a motor move. DCSS forwards this message to all GUI clients as a `gtos_update_motor_position` message.

The format of the message is

```
htos_update_motor_position motorName position status
```

where

- *motorName* is the name of the motor
- *position* is the new position of the motor
- *status* is the status of the motors See Section 9.4.3.

9.4.6 `htos_report_shutter_state`

This message reports a change in the state of a shutter. This may occur as a result of handling the `stoh_set_shutter_state` command or during a timed exposure with automated shutter handling. DCSS forwards this message to all GUI clients as a `stog_report_shutter_state` message.

The format of the message is

```
htos_report_shutter_state shutterName state
```

where

- *shutterName* is the name of the shutter.
- *state* is the new state (open | closed.)

9.4.7 htos_report_ion_chambers

This message reports the results of counting on one or more ion chambers in response to the `stog_read_ion_chamber` message. The first three arguments are mandatory. Additional ion chambers are reported by adding additional arguments. DCSS forwards this message to all GUI clients as `stog_report_ion_chambers` message.

The format of the message is

```
htos_report_ion_chambers time ch1 counts1 [ch2 counts2 [ch3 counts3  
[chN countsN]]]
```

where

- *time* is the time in seconds over which counts were integrated.
- *ch1* is the name of the first ion chamber read.
- *cnts1* is the counts from the first ion chamber.

9.4.8 htos_send_configuration

This message requests that the configuration of the specified device (as remembered by DCSS) be returned to this DHS. DCSS will respond with a `stoh_configure_device` message for the device. This message is not forwarded to the GUI clients.

The format of the message is

```
htos_send_configuration deviceName
```

where *deviceName* is the name of the device for which the configuration information is needed.

9.4.9 htos_simulating_device

This message indicates that the specified device is being simulated by the hardware server. DCSS forwards this message to all GUI clients as the `gtos_simulating_device` message.

The format of the message is

```
htos_simulating_device deviceName
```

where *deviceName* is the name of the device which is being simulated.

Note: If this message is not sent, the BLU-ICE clients will not know what motors are being simulated and what motors are real.

9.4.10 htos_operation_update

This message can be used to send small pieces of data to the GUI clients as progress is made on the operation. It can also be used to indicate to a calling GUI client that the operation cannot continue until the caller performs another task.

The message format is as follows:

`htos_operation_update operationName operationHandle arguments`

- *operationName* is the name of the operation that completed.
- *operationHandle* is the unique value that indicates which instance of the operation completed.
- *arguments* This is a list of return values. It is recommended that list of return arguments adhere to the overall DCS protocol (space separated tokens), but this can only be enforced by the writer of the operation handle.

9.4.11 htos_operation_completed

The message is used to indicate that an operation has been completed by this hardware server.

The general format of the message is

`htos_operation_completed operationName operationHandle status arguments`

where

- *operationName* is the name of the operation that completed.
- *operationHandle* is the unique value that indicates which instance of the operation completed.
- *status* Anything other than a `normal` in this field will indicate to DCSS and BLU-ICE that the operation failed, and this token will become the reason of failure.
- *arguments* This is a list of return values. It is recommended that list of return arguments adhere to the overall DCS protocol (space separated tokens), but this can only be enforced by the writer of the operation handle.

9.5 Server To GUI Client Messages (*stog*)

All commands sent from DCSS to a GUI client start with a *stog* (pronounced s-2-g).

9.5.1 *stog_become_master*

A command telling the GUI client that is now the **ACTIVE** GUI client. No arguments.

9.5.2 *stog_become_slave*

A command telling the GUI client that is not the **ACTIVE** GUI client. No arguments.

9.5.3 *stog_configure_real_motor*

A command to reconfigure a real motor. Arguments are the same as *stoh_configure_real_motor* see Section 9.6.4.

9.5.4 *stog_configure_pseudo_motor*

A command to reconfigure a pseudo motor. Arguments are the same as *stoh_configure_pseudo_motor*, see Section 9.6.5.

9.5.5 *stog_motor_move_completed*

Indicates that the move on the specified motor is now complete. Arguments are the same as for *htos_motor_move_completed*, see Section 9.4.3.

9.5.6 *stog_motor_move_started*

Indicates that the requested move of a motor has begun. Arguments are the same as for *htos_motor_move_started*, see Section 9.4.4.

9.5.7 *stog_no_hardware_host*

Indicates the hardware server for the specified device is not connected to DCSS. This message is sent by DCSS when a command from a GUI client refers to a device that cannot be accessed because the associated hardware server is not connected.

The format of the message is

stog_no_hardware_host *deviceName*

where *deviceName* is the name of the device for which no hardware server is connected

9.5.8 stog_other_master

Indicates that another client is currently the **ACTIVE** client. This message is sent to a particular GUI client if it sends a **gtos_become_master noforce** message while another GUI client is master. No arguments.

9.5.9 stog_report_ion_chambers

Reports the results of counting on one or more ion chambers in response to the **stog_read_ion_chamber** message. Arguments are the same as for **htos_report_ion_chambers**, see Section 9.4.7.

9.5.10 stog_report_shutter_state

Reports a change in the state of a shutter. Arguments are the same as for **htos_report_shutter_state**, see Section 9.4.6.

9.5.11 stog_simulating_device

Indicates that the specified device is being simulated by the a hardware server. Arguments are the same as for **htos_simulating_device**, see Section 9.4.9.

9.5.12 stog_unrecognized_command

Sent to a specific GUI client if it sends a DCS message that DCSS does not recognize. No arguments. Useful for debugging.

Note: Sometimes this command is sent when the user is requesting a command that requires the user to be the **ACTIVE** client.

9.5.13 stog_update_motor_position

Updates the position of a motor during a motor move. Arguments are the same as for **htos_update_motor_position**, see Section 9.4.5.

9.5.14 stog_operation_completed

Arguments are same as **stog_operation_completed**, see Section 9.4.11.

9.5.15 stog_operation_update

Arguments are same as `stog_operation_update`, see Section 9.4.10.

9.6 Server To Hardware Messages (stoh)

All commands sent from DCSS to a hardware server (DHS) start with a `stoh` (pronounced s-2-h).

9.6.1 stoh_abort_all

A command to stop all motors and other operations. Arguments are the same as for `gtos_abort_all`, see Section 9.3.1.

9.6.2 stoh_correct_motor_position

Requests that the position of the specified motor be adjusted by the specified correction. This is used to support the circle parameter for motors (i.e., modulo 360 behavior for a phi axis).

The format of the message is

```
stoh_correct_motor_position motorName correction
```

where

- *motorName* is the name of the motor.
- *correction* is the correction to be applied to the motor position

9.6.3 stoh_start_motor_move

This is a command to start a motor move. Arguments are the same as for `gtos_start_motor_move`, see Section 9.3.8.

9.6.4 stoh_configure_real_motor

The format of the message is

```
stoh_configure_real_motor motorName position upperLimit lowerLimit  
scaleFactor speed acceleration backlash lowerLimitOn upperLimitOn  
motorLockOn backlashOn reverseOn
```

where

- *motor* is the name of the motor to configure
- *position* is the scaled position of the motor

- *upperLimit* is the upper limit for the motor in scaled units
- *lowerLimit* is the lower limit for the motor in scaled units
- *scaleFactor* is the scale factor relating scaled units to steps for the motor
- *speed* is the slew rate for the motor in steps/sec
- *acceleration* is the acceleration time for the motor in seconds
- *backlash* is the backlash amount for the motor in steps
- *lowerLimitOn* is a boolean (0 or 1) indicating if the lower limit is enabled
- *upperLimitOn* is a boolean (0 or 1) indicating if the upper limit is enabled
- *motorLockOn* is a boolean (0 or 1) indicating if the motor is software locked
- *backlashOn* is a boolean (0 or 1) indicating if backlash correction is enabled
- *reverseOn* is a boolean (0 or 1) indicating if the motor direction is reversed

This command requests that the hardware server change the configuration of a real motor.

9.6.5 stoh_configure_pseudo_motor

The format of the message is

```
stoh_configure_pseudo_motor motorName position upperLimit lowerLimit
upperLimitOn motorLockOn
```

where

- *motorName* is the name of the motor to configure.
- *position* is the scaled position of the motor.
- *upperLimit* is the upper limit for the motor in scaled units.
- *lowerLimit* is the lower limit for the motor in scaled units.

- *lowerLimitOn* is a boolean (0 or 1) indicating if the lower limit is enabled.
- *upperLimitOn* is a boolean (0 or 1) indicating if the upper limit is enabled.
- *motorLockOn* is a boolean (0 or 1) indicating if the motor is software locked.

A command to reconfigure a pseudo motor at the DHS level.

9.6.6 `stoh_read_ion_chambers`

Requests that one or more ion chambers be read. The first three arguments are mandatory. Additional arguments specify additional ion chambers to read simultaneously. Arguments are the same as for `gto_read_ion_chambers`, see Section 9.3.5.

9.6.7 `stoh_set_motor_position`

Requests that the position of the specified motor be set to specified scaled value. Essentially an optimized `gto_configure_device` for setting position only. Arguments are the same as for `gto_set_motor_position`, see Section 9.3.6.

9.6.8 `stoh_set_shutter_state`

Requests that the state of the specified shutter be set to the specified state. Arguments are the same as for `gto_set_shutter_state`, see Section 9.3.7.

9.6.9 `stoh_start_oscillation`

Requests that an oscillation be performed using the specified motor and shutter, and over the motor range and time interval specified. Arguments the same as `gto_start_oscillation`, see Section 9.3.9.

9.6.10 `stoh_start_operation`

Requests the hardware server to do an operation. Arguments the same as `gto_start_operation`, see Section 9.3.10.

10 Example code listings in ASCII

10.0.1 Test Diffractometer BLU-ICE Script

10.0.2 Test Diffractometer Operation Script

11 Adding new hardware support

Currently, the Blu-Ice/DCS project is rather open-ended regarding the specifics of adding new hardware support. Basically, adding hardware support involves writing a program that translates the Server to Hardware messages described in Section 9.6 into commands issued through the hardware controller's API. Alternatively, the Server To Hardware messages could be translated into another control system's command language, enabling a gateway for DCS to talk to another control system.

The simulated DHS project, (i.e. `simdhs`) is a great place to start when deciding how to implement a new DHS. It handles many of the possible hardware commands in several pages of code. Its main limitation is that it does not have handles for detector data collection related operations.

After studying the protocol, a developer may wish to look at the 'standard' DHS used at SSRL (found in the `dhs` project) which is used for controlling the Galil DMC 2180, ADSC Quantum 4, ADSC Quantum 315, and the MAR345. On the other hand, this code may be too heavy for a starting point, depending on your application. Specifically, the most difficult aspect of this code is that it uses the `mysql` database to acquire its configuration parameters. Developers are often lead to believe that a requirement for a program to be a DHS is that it somehow uses a `mysql` database. This is simple not true, and the '`simdhs`' project is a counter example.

11.0.1 Example: Adding MAR CCD support

The following example lists the basic steps that were used to add support for a MAR CCD using the 'standard' DHS.

1. The MAR CCD protocol was studied and it was determined to be most similar to the existing Quantum 315 protocol. Therefore, it was decided to use the 'standard' DHS as a baseline.
2. The 'standard' DHS uses the `mysql` database to acquire its configuration. Thus, the following table was created to store a MAR CCD's configuration. At the `mysql` prompt:

```

CREATE TABLE MARCCD (
  ControllerID int(11) NOT NULL,
  serialNumber varchar(30) NOT NULL default '0',
  HostName varchar(30) default NULL,
  CommandPort int(11) default NULL,
  DarkRefreshTime int(11) default NULL,
  BeamCenterX double(16,4) default NULL,
  BeamCenterY double(16,4) default NULL,
  darkExposureTolerance float(10,2) NOT NULL default '0.10',
  writeRawImages char(1) default NULL,
  PRIMARY KEY (ControllerID)
) TYPE=MyISAM;

```

3. After the table was created, a sample configuration was inserted into the table:

```

INSERT INTO MARCCD VALUES (1,'0','marpc',3000,180,94.0000,94.0000,0.10,'Y');

```

4. The `dhs_config.cc` file was modified to look at the MAR CCD table when determining its configuration.

```

diff -c -r1.8 dhs_config.cc
*** dhs_config.cc 8 Nov 2002 19:52:38 -0000 1.8
--- dhs_config.cc 13 Dec 2002 18:01:53 -0000
*****
*** 70,75 ****
--- 70,76 ----
    XOS_THREAD_ROUTINE Quantum315Thread (void * parameter );
    XOS_THREAD_ROUTINE MAR345 (void * parameter );
    XOS_THREAD_ROUTINE ASYNC2100 (void * parameter );
+ XOS_THREAD_ROUTINE MARCCDThread(void * paramater);
    #ifdef WITH_CAMERA_SUPPORT
    XOS_THREAD_ROUTINE DHS_Camera (void * parameter );
    #endif
*****
*** 213,218 ****
--- 214,220 ----
    deviceTables["MAR345"] = (XOS_THREAD_ROUTINE_PTR)MAR345;
    deviceTables["CCD"] = (XOS_THREAD_ROUTINE_PTR)Quantum4Thread; //i.e. Quan

```

```
    deviceTables["Quantum315"] = (XOS_THREAD_ROUTINE_PTR)Quantum315Thread;  
+ deviceTables["MARCCD"] = (XOS_THREAD_ROUTINE_PTR)MARCCDThread;  
#endif
```

5. The `dhs_Quantum315.cc` file was copied to `dhs_MARCCD.cc` and the `Quantum315Thread` procedure was modified to `MARCCDThread`.
6. The final step involved coding handles for each operation command and appropriately controlled the MAR CCD.

12 Document Version Information

Working revision: 1.12 Thu Jan 30 01:55:51 2003