AccelNET Control System User Manual

Mark V. Stodola, National Electrostatics Corp.
Richard L. Kitchen, National Electrostatics Corp.

AccelNET Control System User Manual

by Mark V. Stodola and Richard L. Kitchen

Published Not yet published
Copyright © 2005 National Electrostatics Corp.

Abstract

AccelNET, "the Accelerator NETwork control system", is a software package designed for control of electrostatic particle accelerator systems. AccelNET runs on PC hardware under the Linux operating system. This manual covers a broad range of topics including: software operation and familiarization, database design/structure, and database editing.

Table of Contents

Chapter 1. Introduction to Linux

The computer control system consists of one or more personal computers. Each computer runs a copy of the Linux operating system. Linux is a UNIX style operating system that runs on x86 class machines.

If there is more than one computer in the system, they are interconnected by a network. One computer contains the control system software and user files. The rest of the computers access that computer through the network by using NFS. Throughout the rest of the manual, the computer containing the control system software and the user files will be refered to as the file server.

The network may also have terminal servers connected to it. Terminal servers are boxes that have a number of serial ports and a network connection. They allow incoming and outgoing connections to be made between the devices attached to serial ports and the network.

Starting the control system requires that the file server be started first, followed by the rest of the computers and the terminal servers. After all of the computers and terminal servers are up and operating, the accelerator control system may be started.

The accelerator control system is composed of a database manager program and groups of programs that communicate between the database manager and the hardware.

Linux, as well as most other large powerful operating systems, requires an organized startup and shutdown.

When the system is started, various housekeeping programs are loaded and executed. The file system is checked to insure it wasn't damaged when the system was previously turned off. These and other functions are performed in the startup process.

Shutdown is the opposite of startup. Disk buffers that are currently in memory are written back to the disk drive. Housekeeping programs are terminated and the information that has been collected is properly stored. These and other functions are done by the shutdown process.

1. Default Users

AccelNET provides three default users. Each user serves a specific function.

postgres	Administrative/maintanence account. Used for modifications to the AccelNET database.
csadmin	Administrative/maintanence account. Used for starting/stopping the AccelNET services.
csoperator	General user account. Used for operating a running accelerator.

2. Linux Startup Procedure

This procedure explains what to do to start an individual computer. If the system is operating correctly, things should pretty much happen as described. If there are problems, refer to the Linux manuals for more information.

Note that the file server computer should be the first machine started. Remember that the other computers use the file server.

Procedure 1.1. Linux Startup

1. Turn on the power.

   Wait for the disk drive to reach its operating speed and for the machine to perform its power on self tests.

   Wait for the booting menu to appear.

2. Press <Enter> (if you wait long enough the machine will proceed on its own)

Linux may decide to check the file system if it detects the computer was not shut down properly. If error messages appear while the file system is being checked, it indicates that there is a problem with the files contained on the disk. Usually it is safe to answer "y" to the questions fsck (the file system check program) asks. Sometimes the file system check will require a reboot of the machine.

When this sort of a problem occurs, it usually means that the computer was shut off or restarted in a disorganized manner. Perhaps the computer crashed, the power failed, or someone pushed the reset button. Consult the Linux manuals for more detailed information.

When the login prompt appears, the system is ready for operation.

3. Logging into a Workstation

The login name "csoperator" is provided as a default login for accelerator users. It is possible to customize the system to provide an individual login for each user. Please consult the Linux documentation for information on creating user accounts.

When the computer is powered up, it will go through a normal boot process containing messages that are usually followed by [OK]. Once the boot messages are complete, X (the graphical interface) will start up and prompt for a login.

Type in the username and password in the appropriate text boxes. The password field may not be shown on the initial screen, you will be prompted for it later if this is the case. During this time, you also have the ability to select a Window Manager. It is recommended that KDE or Gnome be used. These two selections provide the most user-friendly experience, being similar to Microsoft Windows.

4. Linux Shutdown Procedure

This procedure explains what to do to stop an individual computer. If the system is operating correctly, things should pretty much happen as described. If there are problems, refer to the Linux manuals for more information.

Many Linux desktops have a method to shutting down using the graphical interface. Because the graphical method may vary, the command line method is described here.

Note that the file server computer should be the last machine stopped. Remember that the other computers use the file server.

Procedure 1.2. Linux Shutdown

1. Open a terminal (command prompt).
2. Super-user to root: **su -**
3. Enter the root password.
4. Enter the shutdown command: **shutdown -h now**

   The system will perform a number of operations. If the system does not automatically power down, a message is displayed saying that it is safe to shut off the power.

Note

There are other ways of shutting down, this is just one example. For example, an alternative to the above command might be **poweroff**.

5. Basic Linux Commands

In order to effectively to use the more advanced features of the accelerator control system, the user should be familiar with the process of logging into the system as well as the commands provided by the Linux operating system.

The concepts of pathnames, current directory, and i/o redirection are very important to a good understanding of how the Linux environment works.

Look in the Linux manual pages for descriptions of how these and other commands work.

This is a list of basic commands:

cat	Copy and concatenate files. Used most often to print the contents of a file on the screen. Example: "cat <filename>"
cd	Change directories. Example: "cd <new directory>"
cp	Copy a file. Example: "cp <source> <target>"
ls	List the contents of a directory. Example: "ls -l"
lpr	Print files. Example: "lpr <file>"
mkdir	Make a new directory. Example: "mkdir <directory name>"
pwd	List the current working directory. Example: "pwd"
rm	Remove a file. Example: "rm <file>"

This is a list of more advanced and administrative commands:

tar	File archiving program. (used to copy to and from tape)
fsck	File system check (used when Linux is started).
mkfs	Create a Linux file system.

mount	Mount a disk partition, cd, floppy disk, etc. to be accessed as a Linux file system.
set	List the environment variables (see the sh manual pages).
sh	The shell.
umount	Unmount a file system.
vi	Text editor.

Chapter 2. AccelNET Installation

1. Introduction

This chapter explains the procedures taken to install AccelNET onto a RedHat Linux 9.0 system. All new accelerator control systems shipped from NEC should have AccelNET preinstalled.

Note

Initial setup of RedHat Linux 9.0 is not covered.

2. Preparing the Operating System

To prepare the system you must be logged in, or su'd to root. If you are not root, then **`su -`** and type in the root password.

1. First, check for RedHat installed postgresql and postgresql-server packages.

```
root# rpm -q postgresql-server
root# rpm -q postgresql
```

2. If the packages are installed, you need to remove them.

Note

Be careful of the order you remove the packages. It may complain about dependencies.

```
root# rpm -e postgresql-devel
root# rpm -e postgresql-jdbc
root# rpm -e postgresql-odbc
root# rpm -e postgresql-perl
root# rpm -e postgresql-python
root# rpm -e postgresql-server
root# rpm -e postgresql-tcl
root# rpm -e postgresql-test
root# rpm -e php-pgsql
root# rpm -e postgresql
```

Now that you have postgres removed, you can begin the AccelNET installation.

3. Installation

Continuing on from the Preperation section (as root), you must mount the AccelNET installation media. There are 2 methods of doing this.

Method 1 (AccelNET CDROM):

Note

If /dev/cdrom does not exist, you will need to replace it with the proper CDROM device node.

```
root# mount -t iso9660 /dev/cdrom /mnt/cdrom
```

Method 2 (NFS share on kitchen):

Note

The machine must be properly configured for NEC's internal network and be plugged into the network.

```
root# mount -t nfs kitchen:/acc.base /mnt/cdrom
```

You should now have the AccelNET installation media mounted to /mnt/cdrom. Next, a few libraries must be installed that are not standard in RedHat 9.0.

```
root# cd /mnt/cdrom/redhat/8.0
root# ./install_libs
```

Now that the proper libraries are in place, AccelNET can actually be installed:

```
root# cd /mnt/cdrom/installation
root# ./install_accelnet
```

If all went well, all of the needed files should be located in /AccelNET.

4. Configuration

>

First, set initial passwords for the csadmin and postgres accounts:

```
root# passwd csadmin
root# passwd csoperator
root# passwd postgres
```

Since you are using RedHat 9.0, the camac 'crate_24' command must be overwritten:

Note

Not all of these commands are run as root, pay close attention to the examples! Not all of the chown and chmod commands may be necessary, but are included for completeness.

```
csadmin$ cp /AccelNET/necdrivers/rh9/crate_24 /AccelNET/sbin/crate_24
csadmin$ chown csadmin.cs_admin /AccelNET/sbin/crate_24
```

```
csadmin$ chmod 755 /AccelNET/sbin/crate_24
csadmin$ ln -s /AccelNET/sbin/crate_24 /AccelNET/sbin/crate
```

Next, the nec initialization script must be configured and installed.

```
root# /AccelNET/necdrivers/build_nodes
root# cp /AccelNET/necdrivers/nec /etc/init.d/nec
root# chmod 755 /etc/init.d/nec
root# cd /etc/init.d
root# vi nec
```

1. Uncomment the lines to load and unload the device drivers needed for the specific machine.

   For RedHat 9.0, you will need to insert the 'rh9' directory between the necdrivers and filename in the commands. (e.g. /sbin/insmod -f /AccelNET/necdrivers/rh9/f_ksc2915mod_24.o major=80) Make sure you use the _24.o files.

2. Insert the following line just after the 'chkconfig: 2345 92 92' line:

   # description: NEC AccelNET device driver loading/unloading

3. Save the file and close vi.

4. Finally, add the init script to chkconfig:

   ```
   root# chkconfig --add nec
   root# chkconfig nec on
   ```

It is a good idea to reboot (type 'reboot' as root) at this point to make sure that the nec startup script is working and that postgres is starting properly. After rebooting, you can view the loaded modules by typing 'lsmod' and check for errors with 'dmesg'. To verify that postgres started properly, make sure that 'ps ax | grep postmaster' returns an entry with '/AccelNET/pgbin/bin/postmaster -S -D/AccelNET/postgres' or similar.

Chapter 3. Introduction to AccelNET

An operator interacts with the accelerator control system by using a workstation. There may be one or more workstations on the system. A workstation consists of a computer, keyboard, mouse, and perhaps an assignable meter/knob system. The workstation may also be connected to CAMAC or other types of data acquisition. Any workstation may access any parameter in the system and change the parameter if it has write permission.

1. Machine Parameters in the Control System Database

The control system is made up of devices. Examples of devices are Faraday cups, charging voltage supplies, bending magnets, and electrostatic quads.

A device is identified by what is called the Label. It is 8 characters long. By convention the label is usually (not always) made of three fields.

The first field is the device name and is 3 characters long. The device name is to be right padded with spaces if less than 3 characters are needed.

Examples:

FC	Faraday Cup
EQ	Electrostatic Quad
FOC	Focus Power Supply
TPS	Terminal Potential Stabilizer

The second field describes a region on the beamline.

Examples:

S1	First ion source
01	Preacceleration beamline
TN	Tank entrance area (usually refers to items in the tank)
TX	Tank exit area (usually refers to items in the tank)

The third field is a serial number. It describes an occurance of a device in a region on the beamline.

A complete Label looks like this:

FC 01-1	First Faraday cup on the preacceleration beamline
FC 01-2	Second Faraday cup on the preacceleration beamline
BM 01-1	First preacceleration beamline magnet

EQ TX-1	Post acceleration electrostatic quad (inside tank at exit end)

There are exceptions to this naming convention. For example, the label "SETUP" has as its parameters all of the particle related information such as total machine energy, species, charge state, etc.

A device is made up of a set of parameters. Parameters are considered single control and readback points. These parameters are identified by what is called a tag name. A tag name is a combination of the device label and a RefName.

Example RefNames (typical of a Faraday cup):

PosSC	Position Status Control
PosSR	Position Status Read
CR	Cup Current Read

The complete tags for the parameters associated with first preacceleration Faraday cup are:

FC 01-1 PosSC	Position Status Control
FC 01-1 PosSR	Position Status Read
FC 01-1 CR	Cup current read

2. Starting/Stopping AccelNET Services

The AccelNET services must be started by the 'csadmin' user. The following instructions assume you have logged in as the 'csadmin' user. All clicks will be a single click unless otherwise noted.

Note

Avoid starting the services up multiple times.

Checks are in place to help prevent this, but are not fool-proof. If you are unsure whether the services are running or not, go through the stopping procedures before starting.

2.1. Services Startup (Menu Based)

1. Launch the AccelNET menu if it is not currently open by double-clicking the "AccelNET" icon on the desktop.
2. Click the menu item labeled "AccelNET".
3. Click the sub-menu item labeled "Start Services".
4. A window will appear asking you if you want to clear the database. Select 'Yes' or 'No.' Clearing the Database while starting services will remove the runtime database from memory and reload it from disk. This can be useful when changes have been made to the on-disk database. It is safe to clear the database when no changes have occured.

If everything worked, a window will appear stating that the services have been started. Otherwise, a window will appear with an error message stating the problem.

2.2. Services Startup (Command Line)

1. If you are not logged in as csadmin, type 'su - csadmin' and enter csadmin's password.
2. To verify the current prompt is owned by csadmin, use the 'whoami' command.
3. Enter the command 'dbstart' to start the database manager.
4. You will be asked if you want to clear the database. Answer the question with 'y' or 'n'. Clearing the Database while starting services will remove the runtime database from memory and reload it from disk. This can be useful when changes have been made to the on-disk database. It is safe to clear the database when no changes have occured.
5. Enter the command 'startio' to start the AccelNET tasks.

If everything worked, no error messages will be displayed and the command prompt should be visible again.

2.3. Services Stop (Menu Based)

1. Launch the AccelNET menu if it is not currently open by double-clicking the "AccelNET" icon on the desktop.
2. Click the menu item labeled "AccelNET".
3. Click the sub-menu item labeled "Stop Services".
4. A window will appear asking you if you really want to stop the services. Select 'Yes' or 'No'.

If everything worked, a window will appear stating that the services have been stopped. Otherwise, a window will appear with an error message stating the problem.

2.4. Services Stop (Command Line)

1. If you are not logged in as csadmin, type 'su - csadmin' and enter csadmin's password. To verify the current prompt is owned by csadmin, use the 'whoami' command.
2. Enter the command 'cskill' to stop all services.
3. You will be asked if you want to kill the database server. Answer the question with 'y'.

If everything worked, no error messages will be displayed and the command prompt should be visible again.

3. Using the AccelNET Tools

All of the tools listed here may be started from the AccelNET menu.

Xcrt	The page display program. See the section titled "Introduction to Xcrt" for more information.
gvm	Creates a small window containing the GVM readback in a large font size. It provides a method for placing the terminal voltage readback permanently on the screen.
bpm	The BPM assignment program.
Xerrlog	An error logging program that displays messages sent from other parts of AccelNET such as the interlock manager. AccelNET may be configured (via the database) to provide error messages for many types of events.
Print Params	Prints out a list of the machine parameters. Clicking the menu entry invokes a submenu containing the choices of what to log. The submenu contains ion source and target beamline pairs. For example, clicking on an entry named "log S1-->05" would cause all of the parameters from source 1 to beamline 05 to be logged.

4. Introduction to Xcrt

The graphical user interface for AccelNET is made up of two programs: crt and ts. The operator interacts with crt while ts runs in the background handling events. From this point on, these two programs will be referred to as Xcrt.

4.1. Launching Xcrt (Menu-Based)

1. Launch the AccelNET menu if it is not currently open by double-clicking the "AccelNET" icon on the desktop.
2. Click the menu item labeled "AccelNET".
3. Click the sub-menu item labeled "Control Pages".
4. Click the sub-menu item labeled with the page you wish to display.
5. A window should appear showing you the page you selected.

4.2. Launching Xcrt (Command Line)

The following are commonly implemented commands to open Xcrt.

necclients crt	Start a predefined set of windows (visual display, table of contents, and help page)
necclients one_crt	Start a single predefined window (visual display)

4.3. Organization of the Xcrt Display Window

The Xcrt display window is divided into three sub windows. The windows are called: the mouse window, the keyboard window, and the page window.

Example Xcrt Display.

4.3.1. Mouse Window

The mouse window consists of two display lines. It is used to show the readback and control parameters currently assigned to the mouse. In general, the upper line is the readback, lower is the control.

Selected parameters occur in pairs (readback and control point).

For example, when the charging power supply voltage control is selected, the selected readback is the positive power supply voltage readback.

The association between control points and readback points is determined when the page is constructed and may vary between pages.

A parameter is selected by placing the cursor on the numeric value or icon of the desired parameter and pressing the select button on the mouse.

There may be more than one parameter set located under a field. An alphanumeric field has a possible two parameter sets. An icon field has a possible four parameter sets. Clicking on the selected parameter multiple times cycles through the sets.

Note that some parameter sets may contain just a readback value, some may have just a control value, some may have both, and some may have both fields set to the same parameter.

4.3.2. Keyboard Window

The keyboard window consists of two display lines. The upper line is used to type commands. The lower line is for displaying error messages. Error messages are normally displayed in red. Pressing enter (<cr>) clears the error message.

4.3.3. Page Window

The page window displays the currently selected page.

The display is organized as a set of pages. The pages are organized by machine region. A separate page is provided for each injector, a page for the low energy beamline, etc. The pages are arranged to provide overlap between machine regions (the same parameter may appear on a number of pages). For example, the injector pages have some of the low energy beamline components on them. This is done to minimize the number of page changes required when tuning beam.

Information on the pages is provided in two manners.

Alphanumeric	Parameter name, value and units
Icon	Picture of a device such as a Faraday cup where the color represents the status

4.3.4. Display Colors

Alphanumeric pages

Green	Value is within database defined limits.
Red	Value is outside of database defined limits.
Violet	Indicates a status error. A CAMAC error such as a missing module. A DUTEC error such as a communication problem or lack of power.

Icons

Dark Blue	Device is inactive (power off).	
Green	Device is active (power on).	
Yellow	Various meanings depending upon the icon.	
	Faraday Cup	Cup is inserted into the beam path
	Double Slit	Slit is in motion.
	TPS Corona Probe	Probe is moving.
	GVM and icons inside of tank	Indicates TPS operating mode, gvm icon yellow means gvm mode.
Red	Indicates an error condition. Example: control system has told Faraday cup to go out and cup status read indicates cup is in	

4.4. Basic Xcrt Operation

4.4.1. Using the Mouse

The mouse may be placed in three different operating modes: inc/dec, rollerball, and x/y. The mice have three buttons that are used in different ways depending on the mode selected. The current mouse operating mode is selected via keyboard command. The operating mode is indicated by the prompt at the beginning of the command line, located in the keyboard window.

The mouse operating mode is displayed by the command line prompt as follows:

id/f>	Indicates that the mouse is in fast inc/dec mode.
id/s>	Indicates that the mouse is in slow inc/dec mode.
rl/f>	Indicates that the mouse is in fast rollerball mode.
rl/s>	Indicates that the mouse is in slow rollerball mode.
xy/f>	Indicates that the mouse is in fast x/y mode.
xy/s>	Indicates that the mouse is in slow x/y mode.

Mouse buttons operate in the following manner:

4.4.1.1. Inc/Dec Mode

A parameter is selected with one of the buttons while the other two are used to increase and decrease the selected parameters value.

Left	Select a parameter
Middle	Decrease the selected parameter value
Right	Increase the selected parameter value

4.4.1.2. Rollerball Mode

A parameter is assigned to the X axis of the mouse. One button is used to assign parameters. The mouse is "armed" by pressing the other button. While armed, rolling in the X axis changes the parameter value.

Left	Arm selected parameter while pressed. Decrease/Increase selected parameter value by moving along the X axis.
Middle	Select a parameter
Right	Note used

4.4.1.3. X/Y Mode

A parameter can be assigned to either of the mouse axes. Two of the buttons serve to assign parameters. The mouse is "armed" by pressing the remaining button. While armed, rolling around changes the parameter values.

Left	Arm selected parameters while pressed. Decrease/Increase selected parameter values by moving along the X and Y axes.
Middle	Select the X axis parameter
Right	Select the Y axis parameter

The commands "xy", "rl", "id", "ms" and "mf" are used to set the mouse operating mode. Please see the section on keyboard commands for more information on their use.

4.4.2. Keyboard Commands

The following is a list of basic commands available through the keyboard. For a complete list of commands, refer to page 2 within Xcrt.

pg n	Change to page 'n' in the window from which the command is entered.
ch	Change the value of the presently assigned parameter. If the current mode is inc/dec, the syntax is "ch <value>". If the current mode is x/y the, syntax is "ch <Xvalue> <Yvalue>".
sv	Save the value of the presently assigned parameter. If the current mode is x/y, both parameters are saved.
rs	Restore the previously saved value of the presently assigned parameter. If the current mode is x/y, both parameters are restored.
lmr	List readback limits of the presently assigned parameter. If the current mode is x/y, both parameters are listed.
lmc	List control limits of the presently assigned parameter. If the current mode is x/y, both parameters are listed.
ldc	List the contents of the presently assigned control data record. If the current mode is x/y, the command is ignored.
ldr	List the contents of the presently assigned readback data record. If the current mode is x/y, the command is ignored.
valc	List the current value of the presently assigned control parameter. If the current mode is x/y, the command is ignored.
valr	List the current value of the presently assigned readback parameter. If the current mode is x/y, the command is ignored.
id	Enter increment/decrement mode.
rl	Enter rollerball mode.
xy	Enter x/y mode.
mf	Set the mouse fast. Affects the speed of parameter changes via the mouse.
ms	Set the mouse slow. Affects the speed of parameter changes via the mouse.
<cr>	(carriage return) Used to terminate a command. Typing a <cr> causes the command line to be displayed.
<ff>	(form feed - ctrl-L) clear the command line.
<del>	(delete) delete an entered character.

5. Assignable Meters

There may be several assignable meters in the system. Each meter module has a mantissa display (the meter), a liquid crystal display (LCD), two range select buttons, and an assign button. There is also an analog jack for each meter located on the rear of the meter chassis. The analog jack gives the value of the mantissa. The voltage range is 0-10v.

The current meter assignment is shown on a LCD located above the meter. The first line of the display shows the Label and RefName for the assigned parameter. The second line shows the parameter value and units. The third line shows the currently selected meter range.

The meter system is only accessable through the workstation it is connected to. The currently assigned readback parameter in the focused window is assigned to a meter when the assign button for that meter is pushed. If the mouse readback parameter is NULL, the meter is unassigned.

There are two meter range buttons. Pressing a button increases or decreases the meter range.

If both range buttons are pressed at the same time, the meter enters freeze mode. Freeze mode is indicated by the message "frz" on the 3rd line of the LCD. This mode is used with autoranging parameters such as the Faraday cup current reads. Pressing a button increases or decreases the meter range. Pressing both buttons at the same time when in freeze mode exits freeze mode.

Meter overrange is indicated by the word "overrange" on the LCD. No meter assignment is indicated by the LCD being blank.

Certain types of devices can not be assigned to the meters. These usually consist of digital (status/control) parameters. For example: Faraday cup position reads.

6. Assignable Knobs

There may be several assignable knobs in the system. Each assignable knob module consists of a knob, a LCD, two range select buttons, a save button, a restore button, and an assign button.

The current knob assignment is shown on the LCD located above the knob. The first line of the display shows the Label and RefName for the assigned parameter. The second line shows the parameter value and units. The third line shows the current knob sensitivity setting.

The meter system is only accessable through the workstation it is connected to. The currently assigned control parameter in the focused window is assigned to a knob when the assign button for that meter is pushed. If the mouse control parameter is NULL, the knob is unassigned.

The knob sensitivity is given in turns to full scale (tfs). For example, if you are controlling a 15KV power supply and the sensitivity is set to 50 tfs, each complete turn of the knob will increase or decrease the voltage by 300V (15KV/50tfs = 300V). Pressing a range button increases or decreases the knob sensitivity.

The save and restore buttons work in exactly the same way as the keyboard "sv" and "rs" commands. Pressing the save button stores the current value of the parameter. Pressing the restore button recalls a previously saved value.

7. Accelerator Startup Procedures

The accelerator commands have been arranged in a way that hopefully provides an organized and convenient method of starting up the machine. The intention is to start the machine up in stages or layers.

The best order for issuing commands, using S2 and 1B as examples, is:

1. If AccelNET is not already operating, follow the startup procedures elsewhere in the manual.
2. Select "Machine->Master Power->On" from the AccelNET menu. This turns on all of the machine basics like Faraday cup controllers, bending magnets, etc.
3. Select "Machine->Source 2->On" from the AccelNET menu. This command turns the ion source cooling, deck power, and bias supply on. It also closes the source Faraday cup.
4. Select "Machine->Source 2->Warm" from the AccelNET menu. This command presets some of the parameters to arbitrary values.
5. If a previously saved ion source setup is availible, click into a terminal window, change to the appropriate directory, and type: send S2sav This presets the preacceleration beamline and the ion source to the previously saved values.

   If there is a source setup on the menu that you wish to use, then select it from the menu.
6. Select "Machine->Accelerator->Warm" from the AccelNET menu. This turns on the rotating shaft, blower, etc and presets the charging and TPS to some running values that won't spark if the chains are started.
7. If a previously saved machine setup is availible, click into a terminal window, change to the appropriate directory, and type: send MACHsav This sets the TPS and charging system to previously saved values.
8. If a previously target beamline setup is availible, click into a terminal window, change to the appropriate directory, and type: send ??sav (where the name of the appropriate beamline is substituted for "??") This sets the post acceleration components to previously saved values.
9. Open the gas stripper valve SLOWLY until the desired pressures are obtained on the tube entrance and exit vacuum gauges. Keep in mind that the presence of beam changes the vacuum reading.
10. Check the setting of the charging voltage and turn it down if necessary.
11. After the corona probe finishes moving into position, select "Machine->Accelerator->Run" from the AccelNET menu. This starts the charging chains.
12. Open the Faraday cups, increase the charging, etc to get the beam through the machine.

The sources usually require some warmup time, therefore you will probably need to start with a lower charging voltage than the final setting from the last run and increase the charging as the source warms up.

If the charging system is balanced correctly, the beam should go all the way to the target with minimal retuning. The parameters most in need of adjustment are the ion source parameters and the stripping gas. With a little practice, it is easy to return the beam to the target at the same energy previously run and with the same beam current.

The user should resist the temptation to immediately start retuning the machine and instead look for and carefully adjust those items that are the least well controlled by the computer such as gas, charging and ion source focus.

If a previously saved run is not being loaded, the ion species setup information should be entered on the machine setup page.

Chapter 4. AccelNET Console Commands

These are commands which may be used after the control system has been started. They provide an easy to use method for starting up the accelerator equipment.

Most, not all, of the commands listed here have been implemented in the menus. You may type the command in the terminal window or you may invoke the command via the menu.

The command system has been implemented in a way that allows it to be easily customized for an individual operator's taste or for individual site requirements.

machine <arg>

Manipulate basic accelerator systems.

on	Turns on all basic machine items (DS, FC, BM, TPS, etc). It does not turn on the charging system, chains, etc.
off	Turns off all basic items and CH, RS, BLW (in case the operator forgot).
misc	This command is optionally present in the machine script and is used mostly for maintenance.

source <arg1> <arg2>

Manipulate source parameters.

S1 off	Turns source S1 off and resets values to 0.
S1 on	Turns on source S1 and sets defaults, also sets SETUP SrcSel = S1.
S2 off	Turns source S2 off and resets values to 0.
S2 on	Turns on source S2 and sets defaults, also sets SETUP SrcSel = S2.

acc <arg>

Accelerator control.

off	Turns blower (BLW), chains (CH), charging power supply (CPS) and rotating shaft (RS) off.
cold	Turns BLW, CH and CPS off.
warm	Turns BLW, CPS TPS, and RS on. Sets default values for TPS and charging system. If "acc run" command is issued, accelerator terminal will charge up and operate "benignly" (approximately 1 MeV).
run	Turns the chains on.
stop	Turns the chains off.

Chapter 5. Advanced AccelNET Topics

1. Manual Pages

The content of this manual does not cover every command and configuration of the AccelNET system. Manual pages have been created for a large percentage of the commands and other aspects of Accel-NET. To access them, use the **man** command, browse the AccelNET web interface, or use the X manual pages menu item from the AccelNET menu.

2. Environment Variables

Shell Variables

`USER_TREE`	Pathname to the AccelNET tree.
`CONF`	Pathname component to contract specific subdirectories.
`DBMAN_HOST`	Host where dbman/pgman runs.
`DOSE_HOST`	Host where DOSEserv runs.
`LOG_HOST`	Host where printer is served from.
`MBS_HOST`	Host where MBSseqTask runs.

DBman Variables

`DB_SIZE_FILE`	Configuration file for master database size.
`AUTH`	Authorization file.

Xcrt Variables

`DISPLAY`	Host where the Xserver is running.
`ASSIGN`	Host where the assign service is running.
`KNOB`	Host where the knob service is running.
`METER`	Host where the meter service is running.
`TREND`	Host where the trend service is running.
`CRT_SIZE_FILE`	Configuration file for crt database size.

3. Directory Structure

$USER_TREE/sbin	Path name to executables.
$USER_TREE/$CONF/startup	Startup scripts.
$USER_TREE/$CONF/asm	Configuration files for meter/knob/assign/fp4d services.
$USER_TREE/$CONF/config	AccelNET configuration files.
$USER_TREE/db/$CONF/db	Database files.
$USER_TREE/db/$CONF/data	Data files used by individual programs.
$USER_TREE/sys/machcmds	General purpose shell scripts and programs.
$USER_TREE/$CONF/machcmds	Contract specific shell scripts and programs.
$USER_TREE/$CONF/CmdFiles	Command files containing parameters and values for machcmds scripts.
$USER_TREE/$CONF/ScaleLists	Parameter lists for save, restore, and status operations.
$USER_TREE/$CONF/tftpboot	Terminal server startup and configuration scripts.

4. Manipulation of the Accelerator Runtime Database

In order for information to be saved, restored, scaled or otherwise manipulated, an empty directory must first be created in which to work. The Linux command "mkdir <directory name>" is used for this. It is suggested that the directory name be used to identify the particle run. Linux directory names may be up to 255 characters long. For example, "Au+++3.0MeV" could be the name for a triple plus gold run at 3.0MeV particle energy.

Next change into the directory just created by typing: cd <directory name>

Many of the commands described in the previous section use the current directory as a place to read from and write information into. Most of them are also implemented using Linux shell scripts (please see the Linux manual pages for bash). They in turn use the programs "ReqPar" and "dbmod" in order to communicate with the control system.

The program "ReqPar" reads from a list of tag names and asks the control system for the current values. The values obtained are written into a disk file along with the tag name.

The program "dbmod" may be used to send values to the control system from files created by ReqPar. The files containing the lists of parameters to obtain from the control system and the lists of parameter values obtained are text files and may be created or edited by a standard text editor such as vi.

4.1. Saving the Entire Accelerator Runtime Database

This is a save of all of the tables in the database. Each table is stored in a seperate disk file. The disk files have as their names "<table name>.sav" The commands "up" and "updaemon" are used to save the tables. The command "down" is used in place of "dbload" at startup time in order to reload the saved database. The format of the command is: down all sav

Saving the complete database is most useful in cases where it is desirable to restore the system to a running state without going through a cold start process. For example, if the computer crashes in the middle of a run.

The program "up" is a voluntary save. The operator enters the program and a menu appears asking which tables to save. The usual method is to save of all the tables. The program may then be exited or left running to resave tables when the operator desires. Usually only the DATA table will need to be resaved.

The second program "updaemon" is a one minute interval DATA table save. When the program is invoked, all of the tables are saved. After that, the DATA table is resaved at one minute intervals until exited. The program is exited by pressing "ctrl-C".

A listing of the saved database may be obtained by using "listit".

4.2. Saving a Particle Run from the Accelerator Runtime Database

A particle run is saved by retrieving the values of individual parameters from the runtime database. The saved parameter values are stored in disk files and may be sent to the control system later in order to recreate a run of the same type. They may also be processed by the scaling program to create a run for another energy or ion species.

The program "request <arg>" is used to obtain lists of values from the running database.

The program "send <arg>" is used to send previously obtained lists of values to the running database. The machine parameters are organized by sections of the machine. For example, typing "request S2sav" obtains a list of all values from source 2 to the entrance of the machine.

One peculiarity to note in regard to the bending magnets is that when using S1sav, S2sav, etc as arguments to "request," the field strength setting parameters for the bending magnets must be set to agree with the field strength reading from the hall probe. The field strength setting is sent to the control system when the file is sent, thus invoking the magnet tune routine.

When the machine scaling parameter lists are requested (S1, S2, 1B, etc), the hall probe reading is retrieved and the field setting is generated, therefore this is not a problem.

The program "extract <arg>" may be used to obtain a parameter list from a database previously saved by "up" or "updaemon".

The recommended procedure for saving a particle run using S2 is:

1. Make a directory to hold the run.
2. Change to the directory just created.
3. Check to make sure the magnet field settings match the hall probe readings.
4. Check to make sure selected ion source, selected beam line, particle name, input mass, output mass and charge state have all been set.
5. Check to make sure the double slit position settings match the position readbacks and set if necessary.
6. If running in slit mode check to make sure TRV is set to match GVM.
7. Type "request S2sav"

8. Type "request MACHsav"

9. Type "request 1Bsav"

10. Type "request DS"

It might be helpful to use "vi" to make a file containing notes about the run. For example, the stripper gas and source heater settings.

It also might be helpful to use "up" or "updaemon" to save all the tables. This can be helpful later if the run needs to be scaled, someone wants to look at an odd parameter value, or get a listing.

5. Using the Accelerator Scaling Program

The scaling program allows the particle energy, particle species, and other such parameters to be changed. The current values of the machine parameters can be retrieved from the running database, changed, and sent back to the running database. Alternately, parameters saved from a previous run may be scaled and sent to the running database.

The preacceleration components may be changed seperately from the post acceleration and machine components, thus allowing the injection energy to be changed to obtain a "match" into the accelerator.

Preacceleration and ion source parameters from a run may be combined with post acceleration parameters from another run in order to create a run on a new target beamline.

The scaling process is divided into three parts:

1. Obtaining the current parameter values.
2. Scaling the values.
3. Sending the newly scaled values to the running database.

The scaling process uses parameter lists to do its job. A parameter list is a disk file with lines of text containing the tag name and value of the parameters. The scaling program reads the parameter list file and outputs a new file with the same format containing the newly scaled values.

Parameter lists to be scaled may be obtained from three sources:

1. A retrieved list from the running database using the "request" command.
2. A previously retrieved list from the "request" command.
3. A previously saved database from either the "up" or "updaemon" command. In order to use this source, the "extract" command must be run to convert the saved database to the proper format.

When the scaling program is invoked, a copy of the entire database is read into memory. As each parameter in the list is processed, a scaling key is looked up in the database. The scaling key tells how to process the parameter to obtain the new value.

The scaling program must be provided with information that tells it the current particle energy, mass, and other needed values in order to function properly. This information is read in from a file called "MACHval" when the program is invoked. The file is obtained from the running database by typing "request MACH" or from a database save by "up" or "updaemon" by typing "extract MACH". Either way, it must be present in the current directory for the program to run correctly.

The program changes parameters such as the charge state, mass, and particle energy via a menu.

The accelerator is divided into three regions: preacceleration, machine, and post acceleration. Each region has a particle energy, charge state, and mass associated with it.

The energy relationship between the areas is TotalPartE = InjPartE + MachPartE. The scaling program allows any one of the three variables to be manipulated in the following way:

TotalPartE	If a new value is entered, InjPartE remains constant. MachPartE = TotPartE(new value) - InjPartE
MachPartE	If a new value is entered, InjPartE remains constant. TotPartE = MachPartE(new value) + InjPartE
InjPartE	If a new value is entered, MachPartE remains constant. TotPartE = MachPartE + InjPartE(new value)

The preacceleration region contains all of the beamline components from the ion source to the entrance of the accelerator. When components in this machine region are scaled, the values of all components affected by a change in InjPartE are recalculated. This includes the bending magnets, einzel lenses, and steerers. Also, the cathode, extractor, and deck bias for a SNICS source. For an RF source, the focus, extractor, and deck bias.

The machine region contains the components needed to set the machine to a given terminal voltage. When components in this machine region are scaled, the values of all components affected by a change in MachPartE are recalculated. This includes TRV and corona probe position. The charging voltage is reset to the value needed for no beam operation if it is included in the list to be processed. (The charging voltage is not currently in the parameter lists, therefore it should be reset by hand).

The post acceleration region contains all of the beamline components after the machine region. When components in this machine region are scaled, the values of all components affected by a change in TotPartE are recalculated. This includes all post acceleration optics and the in tank quads and steerers.

The TRV setting is reset by changes in MachPartE. In turn, when TRV is changed, the corona probe position is recalculated. If TRV is changed by a large amount upward, it may be wise to put in a preacceleration cup and turn off the chains before sending the new values. Wait for the corona probe to reach its new position before turning the chains back on. This prevents unnecessary sparking while the probe is moving to the new position.

The field strength of the magnets is calculated on the basis of MfieldR (the current hall probe value) and the new value is MfieldC (the magnetic field setting value). When the new MfieldC values are written into the running database, the magnet autotune routine is invoked causing the magnets to be retuned to the new field strength.

InjPartM is used to calculate preacceleration optics. TotalPartM is used to calculate post acceleration optics.

The runtime system calculates:

InjPartE = InjPartV * abs(InjChgState)

MachPartE = (GVM * (OutPartM / InjPartM) * abs(InjChgState)) + (GVM * abs(OutChgState))

TotalPartE = InjPartE + MachPartE

InjChgState, OutChgState, InjPartM and OutPartM are entered by the operator as part of the machine setup information.

This is an example of how to do a scaling run. We are going to reduce TotalPartE by changing the terminal voltage. Ion source S2 and beamline L6 are being used for the run.

The initial conditions are:

1. TotalPartE = 12.2 MeV
2. InjPartE = 0.055 MeV
3. MachPartE = 12.145 MeV
4. OutChgState = 3+
5. InjPartM = 197
6. OutPartM = 197
7. TRV = 3.036 MV

The steps are:

1. Make an empty directory.
2. Change to the directory just created.
3. Check to make sure particle mass, ion source selection, etc have been entered on the machine setup page.
4. Type "request MACH" to obtain the current machine operation values.
5. Type "request L6" to obtain the current post acceleration component values.
6. Type "scale L6" to enter the scaling program and scale the post acceleration components.
7. Type "a" to select TotPartE from the menu.
8. Type "12.0<cr>" to enter the new value.
9. Type "r" to scale the L6 parameter list.
10. Type "x" to exit the scaling program.
11. Type "scale MACH" to enter the scaling program and scale the machine components.
12. Type "r" to scale the MACH parameter list.
13. Type "x" to exit the scaling program.
14. Type "send MACH" to send the new machine operation values.
15. Type "send L6" to send the new post acceleration component values.

Note

It is not necessary to reenter the desired particle energy after the first invokation of "scale". When "scale" is exited, a file named "scaenv" is written into the current directory. This file contains the setup information for the scaling program. If the file is present, it is reloaded the next time "scale" is invoked.

If the scaling process is repeated by using "request" to obtain new values from the running database, "scaenv" should be removed before invoking "scale".

This is another example of how to do a scaling run. In this example we are going to reduce the injection energy by 5 KeV. The initial conditions are the same as the last example.

The steps are:

1. Make an empty directory.
2. Change to the directory just created.
3. Check to make sure particle mass, ion source selection, etc have been entered on the machine setup page.
4. Type "request S2" to obtain the current preacceleration component values.
5. Type "request MACH" to obtain the current machine operation values.
6. Type "request L6" to obtain the current post acceleration component values.
7. Type "scale S2" to enter the scaling program and scale the preacceleration components.
8. Type "c" to select InjPartE from the menu.
9. Type "0.050<cr>" to enter the new value.

10. Type "r" to scale the S2 parameter list.
11. Type "x" to exit the scaling program.
12. Type "scale L6" to enter the scaling program and scale the post acceleration components.
13. Type "r" to scale the L6 parameter list.
14. Type "x" to exit the scaling program.
15. Type "send S2" to send the new preacceleration component values.
16. Type "send L6" to send the new post acceleration component values.

6. Terminal Server Configuration

This is the setup procedure for the terminal servers.

1. Turn on power and wait for unit to boot.
2. Log into the unit and become the super user.
   a. `Username>` **`kitchen`** (this name doesn't matter)
   b. `Local_1>` **`su`**
   c. `Passwd>` **`system`**
3. Return to boot program.
   a. `Local_1>>` **`init noboot`**
   b. Wait for boot program to come up.
4. Restore factory defaults.
   a. `Boot>` **`flush nvr`**
   b. Wait for reboot.
5. Perform initial configuration.
   a. `Username>` **`kitchen`**
   b. `Local_1>` **`su`**
   c. `Passwd>` **`system`**
   d. `Local_1>>` **`def server ipaddress 192.168.2.6`**
   e. `Local_1>>` **`def server startupfile "192.168.2.1:/tftpboot/contract-t1.strt"`**
   f. `Local_1>>` **`init noboot`**

6. Prevent rarp and bootp.

   a. `Boot>` **`set server bootp disable`**

   b. `Boot>` **`set server rarp disable`**

   c. `Boot>` **`init 451`**

   d. Wait for reboot.

7. System should now be ready to use.

Chapter 6. Accelerator Mass Spectrometry

1. Introduction to AMS

AccelNET provides a complete AMS data collection system capable of managing data collection for any species of interest. All collected information is logged to disk. The system may be reconfigured by operator command to change ion species.

AMS data logging is comprehensive. Data for every multicup (stable isotope off-axis Faraday cup) are logged in disk files and time stamped with the jumping cycle number in which the measurement was taken. Event data for each detected rare isotope particle is logged and is time stamped with the jumping cycle number and event number within the jumping cycle.

A "setup" file is written containing other information such as the terminal voltage, rare isotope, charge state, etc. for each measurement.

All of the logging files are written in ASCII to make them easy to view, manipulate and input to other programs.

As each cathode measurement is completed, a summary of the measurement is written to a disk file and displayed on the screen.

When the runlist is completed (all cathodes are measured), a summary file is written containing the average currents, rare isotope counts, average ratios, standard deviations etc. for all cathodes in the runlist. Other summary files may be created containing tab separated fields suitable for use by Quattro and Excel.

After all measurements for a list of cathodes have been completed, the datasets gathered may be reanalyzed off line if necessary. Parameters such as the rare isotope gates may be changed and the datasets reprocessed to obtain new summaries.

The summaries are then analyzed to perform normalization, background subtraction and so on.

NEC has written software using the PV-Wave (a data analysis package) language which performs both post measurement functions. NEC calls this software IsoNET.

IsoNET is divided into two programs. One program is used for reanalysis (off line data reduction) of the datasets. It traverses a list of datasets specified by the operator, reevaluates them and writes new summary files.

The second program performs normalization, background subtraction and calculates conventional AMS results and uncertainties.

2. Sequential Beam Injection system

The sequential beam injection system consists of an ion source, an electrostatic spherical analyzer, a bending magnet with an insulated chamber, three off axis Faraday cups and an X/Y steerer.

The post acceleration portion of the system consists of a bending magnet, three off axis Faraday cups, an electrostatic cylindrical analyzer and a gas ionization chamber. The figure titled "AMS system components" is a simplified line drawing of the accelerator layout.

Two of the cups at each end of the machine are used for carbon AMS measurements. The two cups plus a third set of cups can also be used for other species of AMS.

The ion source may be either a multicathode solid SNICS source or a multicathode gas SNICS source. In both cases a negative ion beam is accelerated from the source and passed through an electrostatic spherical analyzer (ESA). The ESA removes energy tails in the beam, which are produced by the cesium sputtering process. In systems containing two ion sources the ESA is made rotatable to select which ion source is to be used.

The beam then passes through a bending magnet with an insulated chamber. The bending magnet is set to bend 14C to the correct angle for injection into the accelerator. 12C and 13C are bent past the angle needed for accelerator injection into a set of offset Faraday cups which are monitored by AccelNET AMS. A power supply connected to the insulated magnet chamber is controlled by a signal supplied by the sequential injection control electronics. Voltages are sequentially applied to the magnet chamber to inject 12C and 13C.

A set of X/Y steerers follows the bending magnet. The steerers are also controlled by the jumping system. A different set of voltages is applied for each injected species to correct for slight possible differences in beam position and direction which may be produced by the different gap voltages applied to the magnet chamber.

The beam jumping control electronics may also be used for a simultaneous injection system. In this case bending magnet chamber bias and the steerers are not required but an electronic chopper is used to reduce the average intensity of the 12C ion beam to about 1% of its DC value. The data collection process is otherwise unchanged.

3. Sequential Beam Injection System control electronics

Beam jumping waveform and gating signals are generated by a set of CAMAC modules designed and manufactured by NEC. Below is a description of each of the modules which make up the system. Please refer to the figures titled "AMS data collection System block diagram" and "AMS carbon data acquistion waveforms".

3.1. Sequence Controller

The "Sequence Controller" CAMAC module generates the timing information. A single jumping cycle may be as long as 4 seconds but is more typically about 100ms. A 4MHz clock and 24 bit programming registers provide a resolution of 0.25us/step.

The Sequence Controller may be programmed to free run, meaning that the next jumping cycle starts immediately after the previous one ends, or may be line synced, meaning that the next jumping cycle is started at the next zero crossing of the AC power line. The choice of free run or line sync and the polarity of the zero crossing may be set by the user.

Jumping cycles are counted by the computer. At the end of each jumping cycle a LAM (Look At Me) signal is generated by the Sequence Controller, which then pauses until the LAM is acknowledged by the computer. LAM processing may be disabled by the user for diagnostic purposes.

A ribbon cable connects the Sequence Controller to the other CAMAC modules making up the jumping system. This cable carries the timing data to the other modules.

One jumping cycle is made up of several "states". The Sequence Controller outputs a state number which is used by the rest of modules.

The Sequence Controller contains 16 timing comparator registers. There is one comparator register for each state.

At the beginning of each jumping cycle a counter register is initialized to zero. This register is incremented by the 4MHz clock.

The comparator registers are programmed to increment the state number at the appropriate delta T for each state. The value of the comparator register for the current state is compared to the counter register. When the value of counter register is greater than the value of the comparator register the state number is incremented.

The maximum number of states used in a jumping cycle is programmable. The jumping cycle ends when the value of comparator register for the maximum state number to use is exceeded.

Typical carbon AMS is done using 7 states in the following way. See "AMS system components" for more information.

State 0	This is the rest state. When the sequencer is not performing a jumping cycle. i.e. it is waiting for the next AC line trigger or for the LAM to be acknowledged it is in this state. This state may be used to provide a predelay, for example, if line sync is in use one can delay the start of the actual jumping cycle for some amount of time after the trigger.
State 1	Power supply voltages slew to the correct value for injection of the first abundant isotope.
State 2	High energy side measurement of the first abundant isotope.
State 3	Power supply voltages slew to the correct value for injection of the second abundant isotope.
State 4	High energy side measurement of the second abundant isotope.
State 5	Power supply voltages slew to the correct value for injection of the rare isotope (14C). Usually for carbon AMS this means 0 volts on the magnet chamber, other species such as aluminium may require different settings.
State 6	Rare Isotope data collection and low energy side measurement of both abundant isotopes.
State 7-15	Not used.

3.2. Sequenced D/A Converter

A Sequenced D/A converter CAMAC module provides the waveforms needed to drive the jumping power supplies.

Usually three Sequenced D/A Converters are used in a system. One converter drives the injection magnet chamber bias power supply. The other two are used to drive a set of power supplies connected to steerers installed in the beamline after the injector bending magnet.

The Sequenced D/A Converter contains four analog value registers. Another set of registers associates a value register with the state number provided by the Sequence Controller.

If the value registers are assigned in this way:

VCreg0	Abundant isotope #1

VCreg1	Abundant isotope #2
VCreg2	Rare isotope. Usually this value is zero for the magnet chamber power supply but may be something else for the steerers.
VCreg3	Not used.

Then the state numbers from the Sequence Controller would be assigned as follows:

State 0	VCreg2
State 1	VCreg0
State 2	VCreg0
State 3	VCreg1
State 4	VCreg1
State 5	VCreg2
State 6	VCreg2
State 7-15	Not used.

3.3. Gate Generator

The Gate Generator CAMAC module provides various signals used to control the AMS data collection system. The module provides 8 separate outputs. Each output may be individually programmed to be on or off during any state. The outputs are differential to allow them to drive long cables. Usually the outputs are connected to the "Quad Receiver" described below at the destination end of the signal.

Typically two channels are used to clock the low energy transient recorders, two channels clock the high energy transient recorders, and one channel is the rare isotope gate.

See the sections which follow for more information.

3.4. Quad Receiver

The Quad Receiver CAMAC module is a simple module containing four channels of differential receivers and two TTL level outputs for each receiver. The differential receivers are connected to the gate generator outputs. The outputs may be individually programmed by DIP switches inside of the CAMAC module to provide a high true or low true output.

4. AMS dosimetry supervision

AccelNET AMS provides a complete facility to control AMS data collection. A supervisory program controls the changing of the cathode in the ion source and starting and stopping of data collection. The status of the accelerator is monitored, and the program will pause data collection in the event of a problem such as a beamline valve closing due to a vacuum fault. Data collection startup is inhibited in cases where the operator may have failed to open a Faraday cup or a beamline valve or other accelerator problems may interfere with data collection.

The supervisory program may be configured to call other programs at various points in its execution.

For example, one might implement a program to log accelerator parameters and run this program each time a cathode is measured.

The supervisory program operates from a list of cathodes (a runlist) generated by the user. This list specifies measurement parameters including the number of times to measure the cathode, the warmup time, the data collection time and the data collection mode.

The cathodes may be divided into groups within the list and each group measured consecutively or independently. Data may be collected for a fixed amount of time or a fixed number of rare isotope events.

When collecting for a fixed number of events, a time limit is also used to prevent excessive collection time on cathodes which have low count rate or problems such as low current. This mode collects until one of the limits is reached.

The cathode runlist may be traversed in two ways.

One way is to measure each cathode once and then go to the next cathode in the list. When this method is used the cathode list is traversed as many times as necessary until all cathodes in the list have been measured the number of times specified for each cathode.

The second way to traverse the cathode list is to measure each cathode repeatedly until the cathode has been measured the number of times specified in the runlist. When working in this mode another program in AccelNET can be used to analyze the measurements as they occur and make decisions about whether to go on to the next cathode or measure the same one again.

This is typically used in applications where it is desired to make several short measurements of a cathode and after some minimum number of measurements make a decision based on the scatter of measurements whether to measure again or go to the next cathode.

AccelNET provides a program module which makes this decision based on the results of the most recent N number of runs. It is possible to write program modules using other selection criteria.

IsoNET also provides a "best run" selection feature for off line analysis.

5. Abundant Isotope Data Collection

Abundant isotope collection is performed by a set of current amplifiers and CAMAC transient recorders. The current amplifiers are controlled by the computer and may be adjusted over a wide range.

The output of each current amplifier is connected to a transient recorder. A transient recorder is a type of list mode ADC. During each jumping cycle the transient recorders are individually clocked by signals from the gate generator at the appropriate moment to capture the value of the pulsed current waveform.

Usually each transient recorder is clocked once per jumping cycle. The isotope is sent to the offset Faraday cup by the beam jumping system, and a snapshot of the current value is taken and placed in the internal memory of the transient recorder.

Occasionally (approximately every 10 seconds) data collection is paused, and the list of measurements is uploaded to the computer where the information is placed in disk files for later processing.

Each time a block of data is uploaded numbers such as the average current and isotope ratios are updated in AccelNET.

NEC offers an option to this system which allows the transient recorders to be clocked more than once per jumping cycle. Using this option one can capture several datapoints within each Faraday cup current pulse and perhaps better quantify the measurement.

6. Rare Isotope Data Collection

Rare Isotope data collection is handled by a gas filled, delta-E detector, a set of NIM amplifiers and logic modules, a CAMAC peak holding ADC, a CAMAC multichannel counter and a CAMAC list processor.

A list processor is a CAMAC module which performs CAMAC I/O operations independently of the computer. It contains a large RAM so the data from the I/O operations it performed can be stored.

The detector contains up to six plates for measurement of Etotal, dE1, dE2, dE3, dE4, and dE5.

The analog outputs from the NIM amplifiers, a trigger signal generated by the NIM electronics and gated by a signal from the jumping electronics are fed into the peak holding ADC.

A program is loaded into the list processor by AccelNET and the list processor is enabled. Each time the ADC receives the trigger signal from the NIM electronics it performs a conversion. At the end of the conversion the ADC asserts a LAM signal.

The list processor waits for the LAM from the peak holding ADC to trigger it. Each time the list processor is triggered it runs the program which has been loaded into it.

The program performs a number of CAMAC I/O operations to copy the data from the ADC to the list processor and clear the LAM. It then goes to sleep until the next trigger.

Periodically data collection is paused, and the data collected in the list processor's internal memory are uploaded to the computer.

The rare isotope data collection system uses two channels of the CAMAC multichannel counter.

The ADC trigger signal from the NIM electronics is split and is sent to the ADC and to one channel of the CAMAC counter module. Each time the ADC is triggered the counter is incremented. This provides a count of the number of ADC triggers supplied by the NIM electronics. When the data are uploaded from the list processor this register is read and a parameter containing the total number of ADC triggers is updated.

When the uploaded data are processed the number of events contained in the data block is counted and a parameter containing the total number of events processed by the list processor is updated.

If the ADC is already busy performing a conversion when another trigger is received, a particle event will be missed. By comparing the two numbers one can get an idea of the number of missed events (detector system pileup). Normally the number of missed events is very small, usually less than 0.1% of the total number of events.

Another channel of the counter module is used to count the number of jumping cycles which have occurred. It is connected to a signal from the jumping electronics which causes the counter to increment at the end of each jumping cycle. Each time the list processor is triggered the cycle number is read from the counter and placed in the block of event data. This allows individual events to be stamped with the cycle number in which they occured.

All aspects of rare isotope data collection are controlled by configuration files. Parameters such as the number of channels of data read from the ADC can be changed. This makes it possible to use the system with other types of particle detectors and perhaps use the system for other types of event counting.

For example, a solid state detector requiring only one ADC channel could be used. The program loaded into the list processor would be changed to only retrieve data from a single channel. This has the side effect of allowing more events to be stored in the list processor's memory and decreasing the data collection dead time because a smaller number of CAMAC I/O operations need to be performed.

It is also possible to locate CAMAC hardware needed for other types of experiments in the same CA-

MAC crate and load different programs into the list processor for the various configurations.

7. HISTmngr

HISTmngr is a display tool which allows viewing of the histogram, contours, and gates which have been defined for the species of interest. Any of the defined histograms or contours may be viewed and the event gate settings may be changed from this program.

When rare isotope data are collected and uploaded into the computer the individual event data blocks are processed according to rule sets contained in a configuration file.

The format of the rare isotope event file and sets of one and two dimensional histogramming arrays (contours) are specified here. An event gate may be defined for each spectrum.

As each event data block is processed a line of data is written to the event file and the histogramming arrays are updated. One channel of input data may participate in several histograms and contours at the same time.

Chapter 7. System Maintenance

1. Backup the System to Tape

The backup program allows backups to be made of two groups of files.

The first is the "root" group. Backup of this information is required only when changes are made to Linux and the files associated with it. For example, if new users are added to the system.

The second is the "users" group. As new particle runs are added, this information changes and should be backed up. Currently the backup shell script copies all files owned by postgres, csadmin, csoperator, and kitchen to tape.

The restore program is used to copy files from a tape to the disk. When this operation is performed, files are created or modified on the disk as necessary.

The backup and restore programs are shell scripts and are easy to change. As the amount of information contained on the disk drive grows, the shell scripts may have to be changed in order for all of the information to correctly write to and read from tape.

The backup and restore programs both use the program tar to perform their jobs. See the Linux manuals for information on how to use tar.

Use these steps to perform a backup:

1. Log in as or su to root.
2. Insert a tape into the tape drive.
3. Type "backup <arg>". (See below)
4. The program performs the appropriate function.
5. Log out/exit.

The backup program is invoked as follows:

backup <arg>	Invoke the backup program.
init	Prepare a new tape to recieve information or erase an old tape.
root	Backup the root information as explained above.
users	Backup the user information as explained above.

The restore program is invoked as follows:

restore <arg>	Invoke the restore program.
root	Reload the root information as explained above.

users	Reload the user information as explained above.

2. Creating and Writing CD/DVD Images

To aid in the backup of data, NEC generally provides a CD/DVD writer with each control system. This section will cover the basic usage of some scripts to create and write standardized backups of AccelNET releated data. If you would like to use the CD/DVD writer for other purposes, using a graphical application such as xcdroast[1] or K3b[2] is recommended. The scripts provided are wrappers to the following command line tools: mkisofs, cdrecord, growisofs.

2.1. Creating a CD/DVD Image

CDs and DVDs are normally written using what is refered to as an image. An image can be in many different formats depending on the methods used to create it. The most universal of these images is called an ISO and conforms to the ISO9660 specifications.

The typical usage for CD/DVD images is to backup the AccelNET directory tree. The files contained in the AccelNET tree are owned by several different users and groups. Because of this, the root user must be used to create the backup image. The following is a set of procedures to produce what is called an AccelNET snapshot.

1. Login as or **su -** to the root user.

```
csadmin$ su -
Password: <type root's password and press Enter>
root#
```

2. Change to the /cdrecord directory.

```
root# cd /cdrecord
```

3. Use **buildit** to create the image.

```
root# ./buildit snapshot
```

The file /cd.accelnet will be created that can be burned to CD or DVD.

The **buildit** command may also be used to create custom CD/DVD images. For usage information, run **buildit** without any arguments.

2.2. Writing a CD/DVD Image

[1] http://www.xcdroast.org/
[2] http://www.k3b.org/

Once an image has been created, either using the **buildit** or on your own, it needs to be written to a CD or DVD. To accomplish this, the **recordit** is used. The following is a set of procedures to write the snapshot image to a cd.

1. Insert a blank CD into the CD/DVD writer.
2. Change to the /cdrecord directory.

   ```
   csadmin$ cd /cdrecord
   ```

3. Use **recordit** to write the image.

   ```
   csadmin$ ./recordit cd snapshot
   ```

The **recordit** command may also be used to write custom CD/DVD images. For usage information, run **recordit** without any arguments.

Chapter 8. Database Structure

1. General Information

The Postgres[1] database manager is used to construct and maintain the control system database. In addition, it is used to generate the wire lists needed to assemble the CAMAC interfaces.

All of the relevant information needed to build the control system database is entered via SQL[2] into the Postgres database.

After the database is entered into Postgres, each one of the tables needed by the accelerator control system is extracted from the Postgres database via SQL queries. They are then processed by other programs to translate the information into a format used by the runtime system.

This manual gives definitions for the fields in the tables maintained by the Postgres database and how they are related to each other.

A **bold** field name indicates the primary key. In some cases the table is keyed by more than one field. This is called a composite key. An *italicized* field name indicates an alternate (candidate) key. An alternate key is a field that can be used to uniquely identify a record (row, tuple) in a table in place of the primary key. This means that it must remain a unique value for each record, just like a primary key.

2. Data Point Definition Tables

The LabelRec, DescRec and DataRec tables all work together to provide a definition of the data points in the control system.

More information concerning the use of the Label and RefName fields can be obtained in the operators manual in the section titled "Organization of machine parameters in the Control System Database".

2.1. LabelRec Table

The LabelRec table defines a list of valid device names.

Label	Identifier for the device/group. (e.g. FC 01-1)
Name	String describing the device. (e.g. Faraday Cup)
Lcomm	Comments field.
PartNum	Not in use. The intention was to associate a part number for a controller with a label and print the information on the wire lists.

2.2. DescRec Table

This table provides a view of individual parameters for a device. The bit field to be viewed comes from the DataRec table.

[1] http://www.postgres.org/
[2] SQL - Structured Query Language

The current binary parameter values are located in DataRec.DataVal. A record in this table allows the word and the bit field within the word to be located, masked and right justified. Other fields determine how the value obtained is converted to a physical value and formatted for display.

A parameter value is extracted from the runtime database using DataCvt(). The scaled value is checked against the limit fields (PhyMin and PhyMax) and is returned along with an error code that indicates whether the value is within limits.

If the parameter is used for control, a field specifies the method for control input. See the section titled "CtKeys" for more information. A check is made to see if the writer is attempting to write a value within acceptable limits and if the writer has permission to write into the database through this record.

Label	First half of the tag name. (e.g. FC 01-1) Validated by LabelRec.Label.
RefName	Second half of the tag name. (e.g. PosSC) Validated by RefKeys.RefName.
Llabel	Specifies a logical (symbolic) link to another Label.
LRefName	Specifies a logical (symbolic) link to another RefName. Causes the rest of the fields to be ignored and the fields of the linked Label/RefName pair to be used. It is highly recommended that all of the ignored fields be set to the equivalent of nil or 0. This is due to a possible future overriding capability. Using the same power supply for two pieces of equipment might be an example use for this. (e.g. EQ TX-1 PwrSC might link to CPS TX-1 PwrSC)
Units	An eight char field containing the physical units string. (e.g. KV, V, A, G, or T)
DataType	Data conversion key. Validated by Dkeys.DtNme. See the section titled "DataTypes."
CrtKey	Display format key. Validated by CrKeys.CrNme. See the section titled "CrtKeys."
CtlKey	Modify control format key. Determines how a mouse, keyboard, or other input device affects a parameter. Validated by CtKeys.CtNme. See the section titled "CtlKeys."
Owner	Name of the task that owns the record. Validated by OwnKeys.OwnNme. See the section titled "OwnKeys."
WpermD	Write permissions. See the section titled "Write Permissions."

ScaKey	Database scaling key. Used by the auto-scaling program(s).
	Validated by ScaKeys.ScaNme.
	See the section titled "ScaKey."
Message	When the value of DataType field is Ldisp this field specifies a message table.
	Validated by checking MsgKeys.MsgNme.
	See the section titled "MsgKeys."
SpanMin	The minimum physical(hardware) value of the parameter. This is used to calculate M and B.
	See the section titled "BuildMB."
SpanMax	The maximum physical(hardware) value of the parameter. This is used to calculate M and B.
	See the section titled "BuildMB."
PhyMin	The minimum virtual(software) value of the parameter. Used in the runtime database to determine if an attempted write or read is within bounds.
PhyMax	The maximum virtual(software) value of the parameter. Used in the runtime database to determine if an attempted write or read is within bounds.
IncVal	Used in most CtlKeys to determine step(increment) size of a change. In most cases this value is created automatically by the BuildMB process.
	See the section titled "BuildMB."
M	Conversion coefficient 1. In most cases this value is created automatically by the BuildMB process.
	See the section titled "BuildMB."
B	Conversion coefficient 2. In most cases this value is created automatically by the BuildMB process.
	See the section titled "BuildMB."
DRkey	Used to determine how to treat the extracted binary field.
	See the section titled "DRkey Usage."
Size	Size of the extracted binary field in bits.
Offset	Offset into the binary word from which to extract the binary field.
Addr	RecId of the corresponding binary field in the DataRec table.
	Validated by DataRec.RecId.
Dcomm	Comments field.
MBconvKey	Enables automatic calculation of the M and B values.
MBsetIncKey	Enables automatic calculation of the SetInc value.

2.2.1. DataTypes

See datatypes(7) in the AccelNET manual pages for a list of valid datatypes.

2.2.2. CrtKeys

See crtkeys(7) in the AccelNET manual pages for a list of valid crt keys.

2.2.3. CtlKeys

See ctlkeys(7) in the AccelNET manual pages for a list of valid control keys.

2.2.4. Write Permissions

See write_perm(7) in the AccelNET manual pages for an explanation of the write permissions.

2.2.5. ScaKeys

See scakeys(7) in the AccelNET manual pages for a list of valid scaling keys.

2.2.6. DRkey Usage

The DRkey is a one character field that specifies how to interpret the extracted binary field.

The values of the field and their meanings are as follows:

U	Treat the extracted field as an unsigned number.
I	Treat the extracted field as a signed number. The minimum value of the integer is -(2^(size-1)). The maximum value of the integer is 2^(size-1)-1. For example, if size = 12 the integer has a range of -2048 to 2047.
P	Treat the extracted field as in 'I.' Negative values are treated as zero.
N	Treat the extracted field as in 'I.' Positive values are treated as zero.

2.2.7. SpanMin and SpanMax Usage

SpanMin and SpanMax are used to calculate the values of the data conversion coefficients M and B. SpanMin is the physical value that corresponds to the minimum binary value of the data point. SpanMax corresponds to the maximum binary value. DescRec.DataType, DescRec.Size, DescRec.DRkey and DataRec.DTkey all play a role in the process.

Calculation of the coefficients is carried out by a family of programs that traverse the database and calculate the values for all of the records at one time. Each member of the program family is responsible for handling one value of DataType. Not all of the possible values of DataType are supported. See the section titled "BuildMB" for more information.

2.3. Message Tables

The message tables are used in conjunction with the Ldisp datatype. A message is provided in a table for each possible value of the data point. The value of the data point and the message type are used as an index into the table.

The runtime data conversion and formatting process is split into two pieces. The first part is concerned with obtaining the physical value of the data point. The second part is the display formatting.

The goal of the Postgres implementation is to allow the collection of information needed to create and maintain the message tables. This has been done by creating two tables (MsgKeys and MsgTbl) for message table management. These tables also exist in the runtime system.

The DescRec table contains a field (Message) that associates a message list with it. The field is used only with DescRecs that have DataType = Ldisp. DescRecs with other DataType values have "NullMsg" entered into the Message field as a place holder.

The messages to be displayed are entered in the MsgTbl. There should be an entry for each possible value of the data point even if not all values are expected to occur. This is done to prevent any unexpected behavior. Each entry in the table must have a unique value of MsgVal for a given MGown (Postgres enforces this).

During runtime, the Message field from the DescRec table is used as MsgNme/MGown. A value is extracted from the DataRec table using information stored in the DescRec entry. The resulting value is checked against PhyMin and PhyMax and becomes MsgVal. A list of messages is traversed looking for a matching MGown/MsgVal pair to determine which message to use (MsgText). If no match is found, '*overrange*' is used.

The DataVal field from the DataRec table is used as MsgVal to retreive the particular message (MsgText) from the MsgTbl referenced by MGown and MsgVal.

The MsgKeys table allows each message list to be given a name by which it is referred to in the DescRec.Message field. (e.g. "PwrMsgSC", "PwrMsgSR")

Note

It is necessary for the value of CrtKey to be 'Lmsg' to display the messages.

2.3.1. MsgKeys Table

This table defines a message list and its size.

MsgNme	Name of the message list.
MsgSize	Number of messages in the list.
Comments	Comments field.

2.3.2. MsgTbl Table

This table contains the text for all messages in all of the message lists.

MGown	List that this message record belongs to. Validated by MsgKeys.MsgNme.
MsgVal	Index of the message in the list.
MsgText	Message text.

2.4. DataRec Table

The DataRec table defines a record where data is stored. Usually this record is associated with the data acquisition hardware, however, this is not always the case. There are quite a few places in the control system where information is acquired, rescaled, or otherwise processed to obtain new results. Total beam energy calculation or magnet field strength deviation might be examples of this.

There are also many places where a number is read into the system and never written out to any hardware. Examples might be charge state, desired magnet field strength, or desired particle mass.

RecId	A serial number used to identify the record.
DevType	Used by the I/O software to determine the type of hardware I/O transactions to perform in order to obtain data from (or write to) the device. Validated by DevKeys.DevTypeK. See the section titled "DevType."
Driver	Data acquisition group. Validated by DrvKeys.DrvNme.
Crate	First part of the hardware address. In CAMAC this is the crate number that is set on the actual crate. This value is also used for grouping/sorting of reports.
Slot	Second part of the hardware address. In CAMAC this is the slot number. This value is also used for grouping/sorting of reports.
ChanNo	Third part of the hardware address. In a CAMAC A/D converter this is the channel number. The software that builds the I/O transactions for a module interprets this for a given device. This value is also used for grouping/sorting of reports.
Dither	Number of bits to mask off for dither reduction. Used by the analog read service.
DTkey	Format of the DataVal, PrevVal and SaveVal fields. 'I' is an integer field, 'F' is a floating point field.
DataVal	Present/current value.
PrevVal	Previous/last value.
SaveVal	Value saved by operator command.
DTcomm	Comments field.

2.4.1. DevType

See devkeys(7) in the AccelNET manual pages for a list of valid device keys.

2.4.2. Driver

See drvkeys(7) in the AccelNET manual pages for a list of valid driver keys.

2.4.3. DTkey

There are three possible values for DTkey.

F	Floating point value (signed).
U	Integer value (unsigned).
N	Negated integer value (unsigned). This key is rarely used. It simply negates the binary value after it is extracted when converting to a physical value using a DescRec.

3. Button Tables

The button tables (BoxRec and PadRec) have been deprecated and are slated for obsolete status. The original function of these tables was to provide touch screen or other alternative input support.

4. Data Point Definition Key Tables

These tables are used to validate fields of other tables in the database. The following is true unless marked with an asterisk (*). These tables should not be modified by the user. The tables will only change if new functionality is added to AccelNET.

4.1. RefKeys Table *

Table of RefName definitions.

RefNme	The RefName.
RefVal	An integer value for the RefName. Used to sort Desc records in the Label report.
RefDesc	Comments field.
RefAttNme	A more descriptive name.

4.2. DKeys Table

Table of DataType definitions.

DtNme	The DataType name.
DtVal	An integer value for the DataType.
DtDesc	Comments field.

4.3. CrKeys Table

Table of display key (CrtKey) definitions.

CrNme	The display key name.
CrVal	An integer value for the display key.

CrDesc	Comments field.

4.4. CtKeys Table

Table of control key (CtlKey) definitions.

CtNme	The control key name.
CtVal	An integer value for the control key.
CtDesc	Comments field.

4.5. OwnKeys Table

Table of Owner definitions. Owner names correspond to task numbers in the control system.

OwnNme	The owner name.
OwnVal	The task number of the owner.
OwnDesc	Comments field.

4.6. ScaKeys Table

Database scaling keys.

ScaNme	The scaling key name.
ScaVal	The scaling key value.
Comments	Comments field.

4.7. DevKeys Table

Table of device type definitions.

DevTypeK	Device type name.
DevTypeV	An integer value for the device type.
DevDesc	Comments field.

4.8. DrvKeys Table

Table of driver type definitions. The control system is designed in a way that allows data acquisition from more than one type of data acquisition system at the same time. The driver type specifies which data acquisition system to use.

DrvNme	Driver name.
DrvVal	An integer value for the driver type.
DrvDesc	Comments field.

4.9. CmdKeys Table *

Deprecated (part of button tables).

4.10. QueKeys Table *

Deprecated.

5. Display Page Tables

The display page database is made up of several record types.

PgKeys contains the master records for a display page. There is one record in PgKeys for each page in the system.

The rest of the records define display entities. There is one record for each entity on the screen. An entity can be an alphanumeric field or an icon.

5.1. PgKeys Table

This table associates a display page name with a page number.

PgNme	Page name.
PgVal	Page number.
Dflag	Display flag. The value of this field is either 'Y' or 'N'. If the value is 'Y', all of the records associated with the page are displayed unless their individual Dflags are set to 'N'. If the value is 'N', the entire page is inhibited from being displayed.
PgDesc	Comments field.

5.2. CrtText Table

This table defines static text strings in a display page. These strings can be used for comments and other permanent text.

Together PgNme and RecId form the key for this table.

PgNme	Name of the page. Validated by PgKeys.PgNme.
RecId	A serial number used to identify the record.
CurX	X coordinate on display page.

CurY	Y coordinate on display page.
Dflag	Display flag. The value of this field is either 'Y' or 'N'. If the value is 'Y', this record is displayed. If the value is changed while the page is displayed, the field is erased.
Width	Width (in characters) of the string.
FontType	Font setting to display the text. Valid values are: default, big, and user1 through user8.
Text	The string to be displayed.

5.3. CrtFixed Table

This table defines database fields for display.

PgNme	Name of the page. Validated by PgKeys.PgNme.
DFnme	Type of field to display. Validated by DFkeys.DFnme.
RecId	A serial number used to identify the record.
CurX	X coordinate on display page.
CurY	Y coordinate on display page.
Dflag	Display flag. The value of this field is either 'Y' or 'N'. If the value is 'Y', this record is displayed. If the value is changed while the page is displayed, the field is erased.
Width	Width (in characters) of the string.
FontType	Font setting to display the text. Valid values are: default, big, and user1 through user8.
Label, RefName	Name of the parameter to display. If DFnme is set to Label or Name, only the Label must be set. The RefName field may be NULL.

5.4. CrtDCpnt Table

This table defines Display/Control parameter sets for a display page. The first display parameter (DCdisp1_Label, DCdisp1_DrefNme) is the one displayed on the screen at the given coordinates. The DCdisp and DCctl fields work in sets. For example, when a DCpnt is selected on the screen, DCdisp1 will be displayed as the readback parameter and DCctl1 will be displayed as the control parameter. The # in DCdisp# and DCctl# identify the layer number. The first time an item is clicked, the first layer will appear. Subsequent clicks will increment the layer.

PgNme	Name of the page this record is associated with.

	Validated by PgKeys.PgNme.
RecId	A serial number used to identify the record.
CurX	X coordinate on display page.
CurY	Y coordinate on display page.
Dflag	Display flag. The value of this field is either 'Y' or 'N'. If the value is 'Y', this record is displayed. If the value is changed while the page is displayed, the field is erased. However, the field may still be selected by the mouse until a new page is loaded or the same page is reloaded.
Width	Width (in characters) of the string.
FontType	Font setting to display the text.
	Valid values are: default, big, and user1 through user8.
DCdisp1_Label, DCdisp1_DrefNme	First read back parameter.
DCdisp2_Label, DCdisp2_DrefNme	Second read back parameter.
DCdisp3_Label, DCdisp3_DrefNme	Third read back parameter.
DCdisp4_Label, DCdisp4_DrefNme	Fourth read back parameter.
DCctl1_Label, DCctl1_DrefNme	First control parameter.
DCctl2_Label, DCctl2_DrefNme	Second control parameter.
DCctl3_Label, DCctl3_DrefNme	Third control parameter.
DCctl4_Label, DCctl4_DrefNme	Fourth control parameter.

5.5. CrtBut Table

Deprecated (part of button tables).

5.6. CrtICpnt Table

Icon display definition table. ICdisp and ICctl fields are treated in the same way as in the CrtDCpnt table.

PgNme	Name of the page this record is associated with.
	Validated by PgKeys.PgNme.
ICtype	Type of icon to display.
	Validated by IconKeys.IconNme.
RecId	A serial number used to identify the record.
GrpNme	Name of the beamline region associated with this record.

	Validated by GrpKeys.GrpNme.
CurX	X coordinate on display page.
CurY	Y coordinate on display page.
CurX2	Second X coordinate on display page. This field is used to stretch/modify the geometry of the icon. A value of "0" will result in the default geometry for the given plane.
CurY2	Second Y coordinate on display page. This field is used to stretch/modify the geometry of the icon. A value of "0" will result in the default geometry for the given plane.
ICrot	Amount (in degrees) to rotate the icon from its base orientation (right hand co-ordinates). A value of "0" will result in the no rotation.
ICscale	Amount to scale the icon size. The default (and most common) size is 1.0.
Dflag	Display flag. If the value is 'Y', this record is displayed. If the value is changed while the page is displayed, the field is erased. However, the field may still be selected by the mouse until a new page is loaded or the same page is reloaded.
ICicon_Label, ICicon_DrefNme	Data point from which to get the icon color.
ICdisp1_Label, ICdisp1_DrefNme	First Display parameter.
ICdisp2_Label, ICdisp2_DrefNme	Second Display parameter.
ICdisp3_Label, ICdisp3_DrefNme	Third Display parameter.
ICdisp4_Label, ICdisp4_DrefNme	Fourth Display parameter.
ICctl1_Label, ICctl1_DrefNme	First Control parameter.
ICctl2_Label, ICctl2_DrefNme	Second Control parameter.
ICctl3_Label, ICctl3_DrefNme	Third Control parameter.
ICctl4_Label, ICctl4_DrefNme	Fourth Control parameter.
Comments	Comments field.

6. Display Page Key Tables

These tables are used to validate fields of other tables in the database. These tables should not be modified by the user. The tables will only change if new functionality is added to AccelNET.

6.1. DFkeys Table

This table is used to validate entries in CrtFixed.DFnme.

DFnme	Display field name.
DFval	An integer value for the display field.

DFdesc	Comments field.

See DFkeys(7) in the AccelNET manual pages for a list of valid DFkeys.

6.2. IconKeys Table

This table is used to validate entries in CrtICpnt.ICtype.

IconNme	Icon name.
IconVal	An integer value for the icon name.
IconDesc	Comments field.

See iconkeys(7) in the AccelNET manual pages for a list of valid IconKeys.

7. Interlock Tables

Interlocks are used to provide conditional actions on data points. They are usually used to provide software equivalents of hardware interlocks. An example might be to prevent a Faraday cup from moving out of the beamline unless the next beamline valve is open. The beamline valve cannot be opened unless there is good pressure on either side of it. Icon colors are also determined by entries in this table.

Interlocks are defined by first choosing a data point to be acted on. The point chosen is entered into a record in the ChkList table.

An interlock chain is created by defining records in the ChkPoint table. The ChkPoint table defines what data points will affect the action. The way in which the data point is evaluated is determined by the CPtype field.

The ChkAct table looks at the word formed by the entries in ChkPoint to determine what will happen to the data point in ChkList. If the word matches the checked criteria, a new value is returned for the ChkList data point.

The ChkAlarm table is processed in a similar way that ChkAct is processed, but returns an error message to the user, instead of a value to the data point.

7.1. ChkList Table

This table defines the data point to be acted on. When a request is made to modify a data point listed in this table, the chain is started. A timer is started lasting the timeout value (in seconds) of TMOvalue. If the interlock does not match an entry in ChkAct by the end of the timer, the default value (Value) is entered into the data point.

RecId	A serial number used to identify the record.
CLdskey_Label, CLdskey_DRefNme	Data point to be acted on.
Value	Value to write into the data point if there is no match in ChkAct table.
TMOvalue	Amount of time to wait before writing the default value.
Comments	Comments field.

7.2. ChkPoint Table

This table defines the data points from which to collect data for interlock evaluation. The records in this table are used to build an evaluation word. An evaluation word is the composite of each record's result specified for the interlock chain. Each record defines an offset (bit) position in the evaluation word to store the result.

MrecId	RecId of the Chklist entry that this record is associated with. Validated by ChkList.RecId.
RecId	A serial number used to identify the record.
CPdesc_Label, CP-desc_DRefNme	Data point to obtain information from.
CPtype	Type of evaluation to be performed. Validated by CPTkeys.CPTnme.
Offset	Offset in the evaluation word to write the evaluation into.
LimLo	Set lower limit for evaluation in certain values of CPtype.
LimHi	Set upper limit for evaluation in certain values of CPtype.
Comments	Comments field.

7.3. ChkAct Table

The value of the evaluation word is filtered and compared using Mask and Mask2 in this table. The Mask field defines which bits of the evaluation word to look at. The Mask2 is compared to the bits of the evaluation word defined by Mask. If there is a match, Value is written to the data point described in ChkList.DSkey.

MrecId	RecId of the ChkList entry that this record is associated with. Validated by ChkList.RecId.
RecId	A serial number used to identify the record.
Mask	Bits in the evaluation word to be checked. A bitwise 'AND' operation is performed on this field and the evalution word to generate a check word.
Mask2	Bit pattern to be matched. This is compared to the previously generated check word.
Value	Value written to the data point described in ChkList.DSkey if Mask2 is identical to the check word.
Comments	Comments field.

7.4. ChkAlarm Table

This table contains information needed to select an alarm message if the interlock evaluation fails. The same method of processing Mask and Mask2 in ChkAct is used to process ChkAlarm. Instead of a value being written into the data point, a message is displayed to the user.

MrecId	RecId of the ChkList entry that this record is associated with. Validated by ChkList.RecId.
RecId	A serial number used to identify the record.
Mask	Bits in the evaluation word to be checked. A bitwise 'AND' operation is performed on this field and the evalution word to generate a check word.
Mask2	Bit pattern to be matched. This is compared to the previously generated check word.
Message	Error message.

7.5. CPtype Field Usage

See cp_type(7) in the AccelNET manual pages for a list of valid CPtypes.

8. Numeric Processor Tables

The numeric processing system allows simple algebraic manipulation of data points to form a value to be written into a data point. This is used to perform calculations such as total injection voltage.

Just like the interlock tables, the numeric processor works off a chaining system. The data point to be affected is set in the NumList table.

The NumPoint table is used to perform the algebraic operations. The operations are peformed in order based on RecId. NPtype is used to define what algebraic function to perform (e.g. add, subtract, multipy, divide, sin, cos). A chain is started with an initial stored value of 0. The first record usually inserts an initial value using 'load' or 'add' as the NPtype. When a record is processed, the data point specified by NPdskey is multiplied by the Scale field to create the operand. If NPdskey is NULL, the Scale field is used as a constant operand in the calculation. Once the operand is determined, the operation specified by NPtype is evaluated on the stored value and operand. The result is placed in the stored value. This continues until all of the records in the chain have been processed. Once the chain has completed, the resulting stored value is written to the data point specified in NumList.

To facilitate complex algebraic equations, four subtotal values are available. They can be used by appending '_S#' to NPtype, where # is 1 through 4. Once the subtotal has been calculated, it can be evaluated into the main total using 't_add_S#', 't_sub_S#', 't_mul_S#', or 't_div_S#'. When the above evaluations are done with the main total, the subtotal is automatically cleared.

8.1. NumList Table

This table defines the data point to be acted on.

RecId	A serial number used to identify the record.
NLdskey_Label, NLdskey_DRefNme	Name of the data point to write the result into.
Comments	Comments field.

8.2. NumPoint Table

MrecId	RecId of the NumList entry that this record is associated with. Validated by NumList.RecId.
RecId	A serial number used to identify the record.
NPtype	Type of algebraic operation to perform on this point and the accumulated value. Validated by NPTkeys.NPTnme.
NPdskey_Label, NP-dskey_DRefNme	Data point to obtain information from. If these values are NULL, the value of 1 is assumed.
Scale	Value obtained from the database is multiplied by this before evaluation. A good use of this is to invert the sign of the database value with -1. By using NULL for NPdskey, this field can be used as a constant.
Comments	Comments field.

8.3. NPtype Field Usage

See np_type(7) in the AccelNET manual pages for a list of valid NPtypes.

9. Interlock and Numeric Processor Key Tables

These tables are used to validate fields of other tables in the database. These tables should not be modified by the user. The tables will only change if new functionality is added to AccelNET.

9.1. CPTkeys Table

This table is used to validate entries in ChkPoint.CPtype.

CPTnme	ChkPoint type name.
CPTval	An integer value for the ChkPoint type name.
CPTdesc	Comments field.

9.2. NPTkeys Table

This table is used to validate entries in NumPoint.NPtype.

NPTnme	NumPoint type name.
NPTval	An integer value for the NumPoint type name.
NPTdesc	Comments field.

10. CAMAC Interface Wiring Tables

10.1. JackRec Table

This table contains the jack number and jack types for every connector in the interfaces. It is used to validate connector number entries in the RPrecord, Zrecord, and Trecord tables and to specify connector type on some of the reports.

The two fields, JackTypA and JackTypB, allow specification of both halves of the connector set.

JackNo	Name of the connector.
JackTypA	Connector type.
JackTypB	Connector type.
Comments	Comments field.

10.2. RPrecord Table

This table specifies a connection for one pair in a device connector on the interface.

RPdskey_Label, **RPdskey_DRefNme**	Parameter associated with this pair.
RPpair	Pair number of the parameter.
RPjack	Jack this connection is located in.
PinA	First pin in the pair.
PinB	Second pin in the pair.
ColorA	Color of the first wire in the pair.
	Validated by ColKeys.ColNme.
ColorB	Color of the second wire in the pair.
	Validated by ColKeys.ColNme.
SigNameA	Name of the signal associated with the first wire in the pair.
SigNameB	Name of the signal associated with the second wire in the pair.
Buss	Bus to connect to in order to obtain power. If no connection is required, use "none" in this field.
	Validated by BusKeys.BusNme.
BussP1	First bus pin.
BussP2	Second bus pin.
Comments1	Comments field.

Comments2	Comments field.

10.3. Zrecord Table

This table is used to define a connection point to a CAMAC module. Connection points to modules are associated with DataRecs.

ZrecId	RecId of the DataRec this record is associated with.
Zpair	Wire pair number for the RecId.
PinA	First pin in the pair.
PinB	Second pin in the pair.
Zjack	Jack this connection is located in.
Comments1	Comments field.
Comments2	Comments field.

10.4. Trecord Table

This table is used to provide connection pairs between any two jacks in the interface without the use of data points.

Typically this is used where connections need to be made between two points in an interface, but there is no entry in the database that corresponds to the signal.

For example:

1. If there are two interfaces associated with a single CAMAC crate, signals often need to be routed from one interface to the other through the trunk cable. In this case it is used to pass signals on the destination side of the jumper's connection to the device connector.
2. It is used to make the shield connections for the device cables. In this case the destination connector is a ground block inside the interface and there is only one wire in the pair.
3. In cases where signals must be passed from connector to another without involving CAMAC. This can happen when hardware interlocks need to be passed between devices.

The fields in the Trecord table are:

TRindex	Index number used to identify the record.
JackA	"Source" jack. Validated by JackRec.JackNo.
JackB	"Destination" jack. Validated by JackRec.JackNo.

PinA1	First "source" pin.
PinA2	Second "source" pin.
PinB1	First "destination" pin.
PinB2	Second "destination" pin.
ColorA	First wire color pin.
	Validated by ColKeys.ColNme.
ColorB	Second wire color pin.
	Validated by ColKeys.ColNme.
SigNameA	First signal name.
SigNameB	Second signal name.
Comments1	"Source" connection first comment field.
Comments2	"Source" connection second comment field.
Comments3	"Destination" connection first comment field.
Comments4	"Destination" connection second comment field.

11. CAMAC Interface Wiring Information Key Tables

These tables are used to validate fields of other tables in the database. These tables should not be modified by the user. The tables will only change if new functionality is added to AccelNET.

11.1. ColKeys Table

Wire Color Key table. This table is used to validate wire colors entered into RPrecord and Trecord color fields. The color value may be used to sort by wire color.

ColNme	Color name.
ColVal	An integer value for color name.
ColDesc	Comments field.

This is list of valid wire colors that may be entered into the color fields in the RPrecords and Trecord. All of the entries listed appear in the ColKeys table.

BK	Black
BN	Brown
RD	Red

OR	Orange
YL	Yellow
GN	Green
BU	Blue
VI	Violet
GY	Gray
WH	White
n/a	None
SHLD	Shield

11.2. BusKeys Table

The CAMAC interface usually contains one or more power buses. They are used to provide power to drive relays and other such devices. Sometimes the bus is switched to provide interlocking functions. For example, in some machines the tank pressure switch controls power to a bus in the interface. The power from the switched bus is connected to the rotating shaft, gvm power status control, etc. If the tank is under vacuum, power is not applied to the bus. Even if the control system may have told the device to be on, it won't be. The BusKeys table allows the buses to be given names and incorporated in the wire lists.

Note

This table is slated for deprecation/obsolete status and will be replaced by JackRec, but is still in active use.

BusNme	Bus Name.
BusVal	An integer value for the bus name. This field is provided in order to allow an arbitrary sort by bus.
BusDesc	Comments field.

This is a list of common entries for RPrecord.Buss. All of the entries listed should appear in the Bus-Keys table.

none	No bus connection.
C24	Camac 24v pwr supply.
Nlk1	Interlock Bus 1.
Nlk2	Interlock Bus 2.

11.3. JkKeys Table

This table is used to validate connector types entered into JackRec.JackTypA and JackRec.JackTypB.

JKtype	Name of the jack type.
JKval	An integer value for the jack type. This field is provided in order to allow an arbitrary sort by jack type.
JKdesc	Comments field.

12. Report Usage

12.1. Invoking Reports

dbreport <report_type> <report_arg>

Alarms	List the interlock records.
CM	List the CAMAC modules.
key <arg>	List records of <arg> Key table. (e.g. RefKeys)
msg <arg>	List records of <arg> Message table. (e.g. MsgTbl)
Label	List major database fields, sorted by Label, Desc.
RecId	List records of the DataRec table sorted by RecId.
Module	List records of the DataRec table sorted by Module Address.
mb	List the database scaling coefficients.
lim	List the database spans and limits.
JackRec	List records of the JackRec table.
RPrecord	List records of the RPrecord table.
Zrecord	List records of the Zrecord table.
type	List valid values of <arg> for generating wiring reports. These values are contract dependent. Example: 'C1I1' means Crate 1, Interface 1.
Buss <arg>	Generate Buss connection list.
Buss2 <arg>	Generate Buss2 connection list.
Rpanel <arg>	Generate rear panel to Z pin wiring list.
Zpanel <arg>	Generate Z pin to rear panel wiring list.
composite <arg>	Generate a composite of Buss, Buss2, Rpanel, and Zpanel wiring lists.

12.2. Printing Reports

Most reports generate a file with a name of the form <report_type>.lst where <report_type> is the argument to dbreport. All reports are stored in `$LST`, which can be expanded to /AccelNET/pg/`$CONF`/lst/. For example "Label.lst" is the report file from "dbreport Label". The report may be printed by using the command "lp <arg>" where <arg> is the name of the report file.

13. Report Formats

13.1. Label report

The Label report shows an overview the database. The report is sectioned by LabelRec entries and detailed by RefName.

Label	Device name (LabelRec.Label).
Name	Long device name (LabelRec.Name).
LabelRec Comments	Comments (LabelRec.Lcomm).
Ref	Integer value associated with the RefName (RefKeys.RefVal).
RefName	Parameter name (DescRec.RefName).
AttName	Long parameter name (RefKeys.RefAttNme).
Units	Physical units (DescRec.Units).
DataType	Data conversion key (DescRec.DataType).
CrtKey	Display format key (DescRec.CrtKey).
CtlKey	Device control key (DescRec.CltKey).
Owner	Name of the task that owns the record (DescRec.Owner).
ScaKey	Parameter scaling key (DescRec.ScaKey).
WpermD	Write permissions (DescRec.WpermD).
Message	Name of the Message List if DataType=Ldisp (MsgKeys.MsgNme).
SpanMin	Physical value that corresponds to the minimum binary value of the data point (DescRec.SpanMin).
SpanMax	Physical value that corresponds to the maximum binary value of the data point (DescRec.SpanMax).
PhyMin	Minimum physical value for write limit or read back limit checking (DescRec.PhyMin).
PhyMax	Maximum physical value for write limit or read back limit checking (DescRec.PhyMax).
Sz	Size of the binary data field (DescRec.Size).
Os	Offset into the data word from which to extract the binary data

	(DescRec.Offset).
DR	DRkey value (DescRec.DRkey).
Id	RecId of DataRec associated with this parameter (DescRec.Addr/DataRec.RecId).
DevType	Device type (DataRec.DevType).
DrvKey	Data acquisition driver name (DataRec.Driver).
C	CAMAC crate number (DataRec.Crate).
N	CAMAC slot number (DataRec.Slot).
ChNo	Module channel number (DataRec.ChanNo).
DT	DTkey value (DataRec.DTkey).
Desc Comments	Comments (DescRec.Comments).

13.2. RecId and Module report

The RecId and Module reports are the same information sorted two different ways. The RecId report is sorted by DataRec.RecId in ascending order. The Module report is sorted by Driver, C, N, DevType, ChNo in ascending order.

Both forms of the report are a cross reference listing showing which parameters (DescRecs) in the database are associated with data acquisition channels.

RecId	RecId (DescRec.Addr/DataRec.RecId).
DT	DTkey value (DataRec.DTkey).
DrvKey	Data acquisition driver name (DataRec.Driver).
C	CAMAC crate number (DataRec.Crate).
N	CAMAC slot number (DataRec.Slot).
ChNo	Module channel number (DataRec.ChanNo).
Sz	Size of the binary data field (DescRec.Size).
Os	Offset into the data word from which to extract the binary data (DescRec.Offset).
DevType	Device type (DataRec.DevType).
Label	Label of the parameter (DescRec.Label).
RefName	RefName of the parameter (DescRec.RefName).
AttName	Long name associated with the RefName (RefKeys.RefAttNme).

13.3. JackRec report

This is a listing of all of the jack numbers in the system.

JackNo	Jack number (JackRec.JackNo).
JackTypA	Jack type for end A (JackRec.JackTypA).
JackTypB	Jack type for end B (JackRec.JackTypB).

13.4. RPrecord report

This is a listing of all of the records in the RPrecord table organized by jack number and pair number.

Jack	Jack number (JackRec.JackNo).
Pr	Pair number in the jack.
Pin	For each record there are two possible pin numbers listed, one above the other. The top one is the first pin in the pair (RPrecord.PinA). The bottom one is the second pin in the pair (RPrecord.PinB).
Color	For each record there are two possible colors listed, one above the other. The top one is the first color in the pair (RPrecord.ColorA). The bottom one is the second color in the pair (RPrecord.ColorB).
SigName	For each record there are two possible signal names listed, one above the other. The top one is the first signal name in the pair (RPrecord.SigNameA). The bottom one is the second signal name in the pair (RPrecord.SigNameB).
BussP1	First side of the buss connection, if needed (RPrecord.BussP1).
Buss	Name of the bus this signal is passed through, if needed (RPrecord.Buss).
BussP2	Second side of the buss connetion, if needed (RPrecord.BussP2).
Label, Ref-Name	Name of the parameter this connection is associated with (DescRec.Label, DescRec.RefName).
Comments	Comments (RPrecord.Comments1/RPrecord.Comments2).

13.5. Zrecord report

This is a listing of all of the records in the Zrecord table organized by RecId and pair number.

RecId	RecId of the DataRec this record is associated with (Zrecord.ZRecId).
DT	DTkey value (DataRec.DTkey).
Driver	Data acquisition driver name (DataRec.Driver).
C	CAMAC crate number (DescRec.Crate).
N	CAMAC slot number (DataRec.Slot).

ChNo	Module channel number (DataRec.ChNo).
DevType	Device Type (DataRec.DevType).
Jack	Jack number (Zrecord.Zjack).
Pr	Pair number of the RecId (Zrecord.Zpair).
Pin	For each record there are two possible pin numbers listed, one above the other. The top one is the first pin in the pair (Zrecord.PinA). The bottom one is the second pin in the pair (Zrecord.PinB).
Comments	Comments (Zrecord.Comments1, Zrecord.Comments2).

13.6. Rpanel report

This is a listing of all of the connections in the interfaces organized from the device connectors to the Z connectors or other device connectors. This report is customized for each contract.

If a connection is made from one rear panel connector to another (one use of the Trecord), the connection will appear twice in the listing. Once from the first connector, again from the second connector.

At the start of each new connector, the page is ejected and the jack number and jack type are printed.

Label, RefName	If the record being listed is a RPrecord, the parameter name from the database is listed on the first line of the signal pair (DescRec.Label, DescRec.RefName).
	If the record being listed is a Trecord, the Comments1 and Comments2 fields are listed on the first line and the Comments3 and Comments4 fields are listed on the second line.
SigName	For each record there are two possible signal names listed, one above the other. The top one is the first signal name in the pair based on the type of record (RPrecord.SigNameA or Trecord.SigNameA). The bottom one is the second signal name in the pair based on the type of record (RPrecord.SigNameB or Trecord.SigNameB).
Pin	For each record there are two possible pin numbers listed, one above the other. The top one is the first pin in the pair based on the type of record (RPrecord.PinA, Trecord.PinA1, or Trecord.PinB1). The bottom one is the second pin in the pair based on the type of record (RPrecord.PinB, Trecord.PinA2, or Trecord.PinB2).
Color	For each record there are two possible colors listed, one above the other. The top one is the first color in the pair based on the type of record (RPrecord.ColorA or Trecord.ColorA). The bottom one is the second color in the pair based on the type of record (RPrecord.ColorB or Trecord.ColorB).
Pin	First bus pin, if used (RPrecord.BussP1).
Buss	Bus the pair is routed through, if used (RPrecord.Buss).
Pin	Second bus pin, if used (RPrecord.BussP2).
Z conn	Jack number of the connector this pair is connected to (Zrecord.Zjack).
Pin	For each record there are two possible pin numbers listed, one above the other. The top one is the first pin in the pair based on the type of record (Zrecord.PinA, Trecord.PinA1 or Trecord.PinB1). The bottom one is the second pin in the pair based on the type of record (Zrecord.PinB, Trecord.PinA2, or Trecord.PinB2).

RecId	RecId of the DataRec this entry is associated with (DataRec.RecId).
C	CAMAC crate number (DataRec.Crate).
N	CAMAC slot number (DataRec.Slot).
Ch	Module channel number (DataRec.ChNo).
Os	Offset into the data word from which to extract the binary data (DescRec.Offset).
Comments	For each record there are two comments listed, one above the other. The top one is the first comment field, depending on the type of record (RPrecord.Comments1 or Trecord.Comments3). The bottom one is the second comment field, depending on the type of record (RPrecord.Comments2 or Trecord.Comments4).

13.7. Zpanel report

This listing is similar the Rpanel report except that it is from the Z connector point of view. This report is customized for each contract.

Z conn	Jack number of the Z connector this pair is connected to (Zrecord.Zjack).
Pin	There are two possible pin numbers listed for each record, one above the other. The top one is the first pin in the pair (Zrecord.PinA). The bottom one is the second pin in the pair (Zrecord.PinB).
Color	There are two possible colors listed for each record, one above the other. The top one is the first color in the pair (RPrecord.ColorA). The bottom one is the second color in the pair (RPrecord.ColorB).
Pin	First bus pin, if used (RPrecord.BussP1).
Buss	Bus the pair is routed through, if used (RPrecord.Buss).
Pin	Second bus pin, if used (RPrecord.BussP2).
RP conn	Jack number of the connector this pair is connected to (RPrecord.RPjack).
Pin	There are two possible pin numbers listed for each record, one above the other. The top one is the first pin in the pair (RPrecord.PinA, Trecord.PinA1, or Trecord.PinB1). The bottom one is the second pin in the pair (RPrecord.PinB, Trecord.PinA2, or Trecord.PinB2).
Label, RefName	If the record being listed is a RPrecord, the parameter name from the database is listed on the first line of the signal pair (DescRec.Label, DescRec.RefName). If the record being listed is a Trecord, the Comments1 and Comments2 fields are listed on the first line and the Comments3 and Comments4 fields are listed on the second line.
SigName	There are two possible signal names listed for each record, one above the other. The top one is the first signal name in the pair based on the type of record (RPrecord.SigNameA or Trecord.SigNameA). The bottom one is the second signal name in the pair based on the type of record (RPrecord.SigNameB or Trecord.SigNameB).
RecId	RecId of the DataRec this entry is associated with if it is a RPrecord (DataRec.RecId).
Pr	Pair number of the Zrecord associated with this entry if it is a RPrecord (Zrecord.Zpair).

C	CAMAC crate number (DataRec.Crate).
N	CAMAC slot number (DataRec.Slot).
Ch	Module channel number (DataRec.ChNo).
Os	Offset into the data word from which to extract the binary data (DescRec.Offset).
Comments	If the record being processed is a RPrecord, then RPrecord.Comments1 and RPrecord.Comments2 are used. Otherwise Zrecord.Comments3 and Zrecord.Comments4 are used.

14. BuildMB Process

After the database is entered, the data conversion coefficients for all of the data points must be calculated. This is done by invoking the BuildMB family of programs.

The buildmb script should be run after the initial creation of a new database, as well as when one of the following has been changed in the DescRec table: SpanMin, SpanMax, DataType, Size, DRkey, MBconvKey, or MBsetIncKey. If MBconvKey is not set to 'Y', buildmb will ignore the record when calculating M and B. If MBsetIncKey is not set to 'Y', buildmb will ignore the record when calculating IncVal.

Invoke a buildmb function by typing the following (as user 'postgres'):

buildmb <arg>

Lin	Build coefficients for records with DataType = 'Lin'.
NLin	Build coefficients for records with DataType = 'NLin'.
Alog	Build coefficients for records with DataType = 'Alog'.
NAlog	Build coefficients for records with DataType = 'NAlog'.
CVG	Build coefficients for records with DataType = 'CVG'.
CVG_275	Build coefficients for records with DataType = 'CVG_275'.
CCVG	Build coefficients for records with DataType = 'CCVG'.
PVG	Build coefficients for records with DataType = 'PVG'.
TCG	Build coefficients for records with DataType = 'TCG'.
IGCgp	Build coefficients for records with DataType = 'IGCgp'.
LinSetInc	Set IncVals for records with DataType = 'Lin'. See rules below.
NLinSetInc	Set IncVals for records with DataType = 'NLin'. See rules below.
all	Invokes all of the arguments.

Here is a list that explains, by DataType, how buildmb treats a record.

Lin	Constructs coefficients for all cases. IncVal is set to the value of M for records where DataType = 'Lin' and Owner = 'CrtTsk'.
NLin	Constructs coefficients for all cases. IncVal is set to the value of M for records where DataType = 'NLin' and Owner = 'CrtTsk'.
Alog	Constructs coefficients for all cases. IncVal is not set.
NAlog	Constructs coefficients for all cases. IncVal is not set.
BCD	Not supported. When used, coefficients should be entered manually, usually with M = 1.0 and B = 0.0.
TCG *	Sets M and B to the correct values to provide a scaler between 0 and 10 for the conversion routine. IncVal is not set.
CVG *	Sets M and B to the correct values to provide a scaler between 0 and 10 for the conversion routine. IncVal is not set.
CVG_275 *	Sets M and B to the correct values to provide a scaler between 0 and 10 for the conversion routine. IncVal is not set.
CCVG *	Sets M and B to the correct values to provide a scaler between 0 and 10 for the conversion routine. IncVal is not set.
PVG *	Sets M and B to the correct values to provide a scaler between 0 and 10 for the conversion routine. IncVal is not set.
Ldisp	Not supported. When used, coefficients should be entered manually, usually with M = 1.0 and B = 0.0.
Raw	Not supported. When used, coefficients should be entered manually, usually with M = 1.0 and B = 0.0.
IGCgp *	Sets M and B to the correct values to provide a scaler between -10 and 0 for the conversion routine. IncVal is not set.
AlinLog	Constructs coefficients for all cases. IncVal is not set.
NAlinLog	Constructs coefficients for all cases. IncVal is not set.

DataTypes marked with an * are handled by the runtime system in the following way: When the data(X) is processed, M and B are applied to scale the value between the desired range (Y = MX + B). The scaled value is then sent to what is called a piece table. A piece table is a predefined set of steps/ranges. Each range contains an internal set of M and B values. The internal set of M and B are applied using the same equation to the scaled value to obtain the final physical value.

15. Table Translation

The database must be converted to the correct format before it can be used by the control system. The format conversion is done by a family of programs that all have roughly the same structure.

The table to be converted is extracted from the Postgres database by a SQL query and placed in a file. When the table is extracted, a filter is applied to substitute integer values for names on certain fields. For example, in the DescRec table, the name in the DataType field is substituted for the integer value known by the runtime system. In the above example, the Dkeys table is considered an immutable table. This means that it cannot/should not be changed. The table is in place to prevent bad entries in database. The actual integer values are defined in the filter/AccelNET source code. The Dkeys table will only be changed when there is a change in AccelNET that affects it. The result of this is an ASCII file where each line contains a number of fields separated by a field separator (usually a vertical bar '|') and terminated by a linefeed.

A program then translates the created file into the record structure used by the control system. This process involves steps such as converting strings of characters into floating point numbers. The program that does the translation is specifically tailored for the table in question, producing a new result file.

It is presently arranged so that the extracted Postgres tables occupy a directory. The SQL scripts share the directory with the extracted files along with a shell script that is used to translate the tables.

The table translation programs, invoked from a shell script, are in a separate directory. They read from the directory containing the extracted files and write into another directory.

16. Using SQL

As you become familiar with the fields of the database and how to manipulate them through the data entry screens you may want to learn to use the tools the Postgres database manager provides for manipulation of the database.

SQL is a programming language that allows records to be added, deleted, and modified to/from the database. It makes adding large quantities of records (for example, if a new beam line is added to the system) very easy to do. Consult the Postgres manuals for more information.

All of the reports described in this manual are generated using SQL, some of the UNIX tools, and a report written in C. Examine the report scripts provided with the system for examples.

For more information on SQL, visit http://www.postgres.org/docs/ as well as other online or printed documentation.

17. Database Construction Tools

To help facilitate construction and maintenance of the database a collection of tools have evolved over time. This collection is not guaranteed to be complete. It is provided as a convenience to the customer.

Using the tools requires a working knowledge of the database design, SQL, and the UNIX shell. There are many existing script files to serve as examples.

Chapter 9. Editing

1. Introduction

Editing the control system database requires that you understand the database structure as covered in previous chapters.

To effectively edit and apply changes to the runtime system, you will need access to both the csadmin and postgres users. The actual editing is performed by the postgres user. The csadmin user is only needed to perform a restart of the control system. A restart is required for new additions and modifications to certain tables.

For the remainder of this chapter, the contract name 'sample1' will be used.

2. Console Editing

2.1. Starting a new Contract and Database

If you want to edit an existing database, skip this section.

A contract is typically cloned from an existing, similar contract. The following is how to clone a contract named source1 to sample1.

Start by making a complete copy of the contract directory.

```
csadmin$ cd /AccelNET
csadmin$ cp -R source1 sample1
```

Once this has been done, edit /AccelNET/sample1/environ. Set CONF=sample1 at the very least.

The following commands will create the runtime database directory structure.

```
csadmin$ cd /AccelNET/db
csadmin$ make_db_dir sample1
```

Next make a postgres contract directory, setup the editing tools environment, and create a default AccelNET database. The following should be executed as the postgres user.

```
postgres$ cd /AccelNET/pg
postgres$ make_contract sample1
postgres$ make_tools sample1
postgres$ /AccelNET/pg/sys/dbcreate/builddb/dbcreate sample1
postgres$ cd /AccelNET/pg/sample1
postgres$ cp ../source1/senvi .
```

You should now have the start of an AccelNET database.

After the database has been created, verify that /AccelNET/environ is linked to /AccelNET/sample1/environ and that CONF=sample1 is set in the file. Edit the senvi file to set CONF=sample1 as well. Logout and back in as the postgres user for the changes to take effect.

2.2. Retrieving Tables

In order to modify entries in the database, a table or portion of a table must be retrieved from PostgreSQL. To accomplish this task, the **fetch** command will be used.

```
postgres$ cd $TOOLS
postgres$ fetch Label
```

A file called Label.lst containing the LabelRec table is produced. This file may be opened with any unix/linux text editor. Each line of the file contains a single entry of the table. Fields are separated by a vertical tab (|). From this point, fields can be edited and entries may be added.

2.3. Updating Tables

Updating a table will modify existing and insert new entries. It is usually best to work from a freshly pulled table or to clone from an existing database. Once the changes have been made to the text file, save it and return to the prompt. To update/insert, use the **update** command from the tools directory.

```
postgres$ update Label
```

Watch the output for any errors. Errors can be caused by invalid values, incorrect field positions, or table relation conflicts. If an error occurs on an entry, the update or insertion of that entry will not occur.

2.4. Deleting entries

Deleting entries can be one of the most frustrating tasks with an AccelNET database. This is due to the nature of using a relational database and currently available tools. A basic knowledge of SQL is required to delete entries. As the postgres user, connect to the database using the **psql** command.

```
postgres$ psql sample1
```

A welcome message should display giving you a list of help commands and a prompt 'sample1=>' waiting for a command.

For this example, we will delete an entry out of the DescRec table. We are going to attempt to delete the entry for label 'FC 01-1' and refname 'PosSC' from the table. This will only match one entry in the DescRec table because of the primary keys set for the table. It is important to know what is going to match your delete request to avoid unwanted removal of entries.

```
sample1=> DELETE FROM DescRec WHERE Label='FC  01-1' AND RefName='PosSC';
```

Here is where the trouble will most likely start. The majority of the time, an error will be displayed stating that it cannot delete the entry. This is what is called a trigger. A trigger message will look much like this:

NOTICE: can't delete |FC 01-1|PosSC | it is in use by ChkPoint.CPdesc

A quick glance will tell you that you tried to delete FC 01-1|PosSC and that a reference to it exists in the ChkPoint table. To resolve this, you must delete the reference(s) to the parameter in the ChkPoint table. In addition, pay attention to where you are deleting from. In cases such as the interlock tables (ChkList, ChkPoint, ChkAct, ChkAlarm), related tables may need to be modified to refelect the deletion. See the appendix titled "Resolving PostgreSQL Trigger Errors" for a more in depth look at resolving trigger errors.

The easiest way to find which entry contains the offending parameter is to retreive the table to a text file and perform a search. To exit **psql** simply type **\q** and you will be returned to the normal shell prompt. After finding the entries, decide how they should be handled. Some entries, such as those in the crtDCpnt or crtICpnt tables may contain parameters that are still in use. In this case, it is probably best to replace the offending parameter values with NULL|NULL. Another case would be where the parameter is taking up an entry that will no longer be needed. Use **psql** and a DELETE statement to remove the entry. Again, pay close attention to the statement you use to delete the entry so as not to harm the rest of the table. Once you have removed the offending entries, attempt the original DELETE statement. If all of the offending entries in all of the tables have been removed, the delete will be succesful. Repeat the above process until the delete is succesful.

2.5. Database Conversion

Once modifications to the database are complete, they need to be converted to a format the runtime system can use. To accomplish this task, the **dbconvert** command is used. The command **dbconvert all** will convert all of the tables to the runtime format. Individual tables can be converted by specifying them in the same way fetch and update are called. In general, it is best to use the **dbconvert all** instead of converting individual tables. This will decrease the chance of not converting a needed table. The speed of current processors is able to do the operations fast enough.

There are a few special cases that require prior preparation for conversion. The first case is with the DescRec table. The DescRec table has flags that allow for M, B, and IncVal to be calculated. To do this calculation, you can use the **buildmb all** command. The second case is with any Crt table. The Crt tables calculate their location coordinates based on the RecId. This is done using the **convertX** and **convertY** commands.

2.6. Applying Changes

Now that the desired editing has been completed and the database has been converted, the runtime system must be notified of the new changes. There are two scenarios for notifying the runtime system.

The first scenario is the most reliable as well as the easiest. The disadvantage is that the control system must be halted.

1. Clear the runtime database and stop all of the AccelNET clients.

   ```
   csadmin$ dbclear
   ```

2. Load in the new runtime database from the converted files.

   ```
   csadmin$ dbload
   ```

3. Start the AccelNET clients.

   ```
   csadmin$ startio
   ```

The second scenario allows modification of the runtime database without the need to stop the AccelNET clients. Be aware that this method only works on certain tables and fields in the database. As a rule of thumb, this method will not allow addition of new records, changes to primary keys, or changes to fields that are used as pointers.

Here is a fairly complete list of modifiable fields:

Table Name	Fields Allowed
LabelRec	Name
DescRec	Units, DataType, CrtKey, CtlKey, Owner, WpermD, ScaKey, SpanMin, SpanMax, PhyMin, PhyMax, IncVal, M, B, DRkey, Size, Offset, Dcomm, MBconvKey, MBsetIncKey Although the listed fields are modifiable, it may be dangerous to change Size and Offset on a running system.
MsgTbl	MsgText
PgKeys	Dflag, PgDesc
CrtText	CurX, CurY, Dflag, Width, FontType, Text
CrtFixed	CurX, CurY, Dflag, Width, FontType
CrtDCpnt	CurX, CurY, Dflag, Width, FontType
CrtICpnt	CurX, CurY, CurX2, CurY2, ICrot, ICscale, Dflag
ChkList	Value, TMOvalue
ChkPoint	CPtype, Offset, LimLo, LimHi
ChkAct	Mask, Mask2, Value
ChkAlarm	Mask, Mask2, Message
NumPoint	NPtype, Scale

To perform this type of modification, the **dbmodify** command is used as the postgres user. The following is an example using the DescRec table.

```
postgres$ dbmodify Desc
```

In addition to the listed fields above, there are a few things to consider. For example, modifications to the interlock and numeric chains will not be evaluated immediately. Modifications will be evaluated when a watched parameter changes value in the ChkPoint or NumPoint table for the chain. It may be helpful to use the **cache** command within Xcrt to refresh the currently displayed page. This can fix drawing errors while moving icons around the diagramatic display.

3. Graphical Editing

3.1. About pgEdit

pgEdit is a combination of server-side scripts and a Java applet designed to modify an AccelNET database graphically. The goal of pgEdit is to allow someone with less console and/or SQL experience to make simple changes to AccelNET.

Although this application is not yet complete, it is quite functional in its current state. If you encounter problems/bugs with this application, please contact NEC with a description of the problem so that it may be addressed.

All of the commonly edited tables have basic insert, update, and delete abilities. The following commands have also been implemented: **dbconvert all**, **buildmb all**, **convertX**, and **convertY**. The following reports may also be generated and viewed: Module, CAMAC, Alarms, Label, RecId, M&B, Limits, JackRec, RPrecord, and Zrecord.

Note

Because pgEdit is still under development, it may not have been installed and/or configured on your control system. Please contact NEC if you wish to test this software.

3.2. Using pgEdit

TODO: write me!

pgEdit can be accessed using a Java[1] enabled web browser such as Mozilla, FireFox, Safari, Opera, NetScape, or Internet Explorer. The following is a set of directions on how to access the editor.

1. Open your Java enabled web browser.
2. Point the browser to the control system's hostname. If you are seated at the control system, the hostname 'localhost' will work.

   You will be presented with a web page tailored for AccelNET. Across the top will be the contract name, accelerator type, and the wording "AccelNET Control System Home Page." Several menus will be lined down the left side of the page. These menus include accelerator status views, dosimetry information, documentation, and pgEdit.
3. Scroll down the page to the bottom so that the Database Tools menu is visible.

   Database Tools Menu
4. Single-click on "Database Editor."

   You are now presented with a login form.

[1] http://java.sun.com/

Database Login

5. Fill in the fields and click the "Login" button.

   The hostname should have been filled in for you. Click the dropdown menu to select the contract you wish to edit. In most cases, there will only be one option. The 'postgres' username should be used with no password.

   At this point, the Java applet should load.

6. Single-click the "Connect" button to connect to the database.

   The status bar at the bottom will display a message stating it is connected.

7. Single-click the "(Re)Load" button to load the database into pgEdit.

The user interface should be fairly easy to learn. The database table names are listed as a series of tabs along the left side of the applet. Across the top you will find buttons to perform global functions. The following is a list of the buttons and what they do:

Connect(Disconnect)	Connects/Disconnects to/from the Postgres database.
(Re)Load	Retrieves the tables and populates pgEdit with the latest values. This should be clicked after major deletions and updates.
Refresh Rec	Retrieves and populates the currently displayed screen with the latest values. This is helpful if you made a mistake and noticed it before you submit an update.
Update Rec	Updates the database with the currently displayed information. This should be clicked after making any changes before continuing to another parameter/table.
Delete Rec	Removes the currently displayed record from the database permanently.
Insert Rec	Inserts the currently displayed record to the database permanently.

Appendix A. Resolving PostgreSQL Trigger Errors

This appendix explains how to resolve many triggers while deleting from an AccelNET database.

1. LabelRec Table Triggers

The only trigger that should be encountered while attempting to delete from the LabelRec table is one that refers to the DescRec table. Because the Label field of a DescRec entry validates with the LabelRec table, one must remove all entries referencing the label you wish to delete. If you were renaming a label, you probably updated the LabelRec table and used **fetch** and **update** to rename the DescRec entries. By doing this, the DescRec table creates new entries for the Label/RefName pair, leaving the old ones intacted. For example, we want to delete FC 01-2 because the Faraday cup was removed from the system, or simply was named incorrectly.

First an attempt would be made to delete the cup.

```
sample1=> DELETE FROM LabelRec WHERE Label='FC  01-2';

NOTICE:  can't delete |FC  01-1| it is in use by DescRec.Label
DELETE 0
```

As you can see, a trigger was hit in the Label field of the DescRec table. Since we don't want this label at all, delete it from the DescRec table. Attempt to delete from the LabelRec table again as well.

```
sample1=> DELETE FROM DescRec WHERE Label='FC  01-2';
DELETE 7

sample1=> DELETE FROM LabelRec WHERE Label='FC  01-2';
DELETE 1
```

The delete was successful since the trigger has been satisfied.

There is one other variation that may occur. That is, the label might be used in the logical link Label/RefName in the DescRec table. In this case, use **fetch** and **update** to modify the logical link to another parameter or set the values to NULL.

2. DescRec Table Triggers

When deleting entries in the DescRec table, a number of triggers may need to be handled. Logical links to other DescRec entries is one of them. As described in the LabelRec section of this appendix, use **fetch** and **update** to modify the logical link to another parameters or set the values to NULL.

Another common trigger is with the CrtFixed table. In general, entries in the CrtFixed table pointing to the DescRec entry you wish to delete are no longer needed. Use a SQL delete command to delete these records.

CrtDCpnt and CrtICpnt table triggers are more involved. These two tables can have entries that point to more than just a single DescRec entry. Because of this, care must be taken to remedy the trigger. First, use the **fetch** to identify the entry that reference the DescRec entry. Look at each entry and determine if any other DescRec entries will be affected by the deletion of the entry in the Crt table. If no other DescRec entries will be affected, it is safe to delete the record. If other DescRec entries are affected, the typical action is to set the Label/RefName in the offending Crt table entry to NULL.

Triggers may also be caused by the NumList or NumPoint tables. If a trigger is with the NumList table, it is most likely the case that one wants to delete the numeric chain. To do this, find the NumList entry corresponding to the DescRec you wish to delete. Use the RecId from the NumList table to remove based on the MrecId in the NumPoint table. Then remove the NumList entry using the same RecId.

```
sample1=> DELETE FROM NumPoint WHERE MrecId=112;
DELETE 3

sample1=> DELETE FROM NumList WHERE RecId=112;
DELETE 1
```

If the offending record is simply a NumPoint entry, more steps must be taken. Since the NumPoint table relies on sequential RecId numbers to function properly, removing a RecId in the middle of a chain causes a need to shuffle the RecId numbers up. For example, the Label/RefName we want to delete from DescRec is RecId 3 in a list of five NumPoint actions. If the NumPoint record is deleted from the table, a sequence of 1,2,4,5 is left. This is not acceptable. To correct this, change the RecId of the fourth entry to 3, and the fifth entry to 4. After a **fetch** to verify your changes, you will notice that you now have 1,2,3,4,5 again, with 4 and 5 being identical. Delete the fifth record. Attempt the delete on the DescRec entry again.

Much like the triggers from the numeric chains, the triggers for the interlock chains can require extra steps. The same principles apply to deleting an entire chain. Perform a delete based on the RecId from ChkList, using MrecId in ChkPoint, ChkAct, and ChkAlarm. Then perform the delete based on the RecId in ChkList itself.

If the offending record is an entry in ChkPoint, remove and shuffle the RecId as was described for the NumPoint table. After shuffling the RecIds, correct the Offset values for the entries to remove the dead space left by the deleted entry. For example, the previous chain had 5 checks, each of them containing a single bit of offset. The third check was deleted. The offsets of the old RecId 4 and 5 (new RecId 3 and 4) should be changed to 2 and 3 repectively. In addition, the Mask and Mask2 fields in the ChkAct and ChkAlarm tables must be updated to reflect the deletion of the ChkPoint entry.

3. DataRec Table Triggers

The DataRec table triggers are similar to those of the LabelRec in that it will only conflict with DescRec entries. To remedy this trigger, redirect any DescRec entries that point to the Addr of the DataRec you wish to delete. A common practice is to use the **fetch** command with a modified .pqry to extract only the DescRec entries that point to the Addr you wish to delete. Edit the fetched file and set the Addr to 0. This points the DescRec entries to a NULL DataRec. Use the **update** command to push the changes back to the database. Perform your delete statement again.

4. MsgKeys Table Triggers

The MsgKeys table can run into problems when deleting from two types of triggers. The first type will be caused by the MsgTbl table. The second caused by the DescRec table.

It is assumed you wish to remove a message completely and are not attempting to rename a message list. To handle the MsgTbl, simply perform a delete statement on the table name in the MsgTbl table. For example, one might have created a UserSR message for a custom status read that is no longer used. In this case, the following should be done.

```
sample1=> DELETE FROM MsgTbl WHERE MGown='UserSR';
DELETE 2

sample1=> DELETE FROM MsgKeys WHERE MsgNme='UserSR';
DELETE 1
```

The message list may still be in use by a DescRec entry, preventing deletion. If the DescRec entry is no longer needed, delete it. Otherwise, use the **fetch** and **update** commands to point the record to a different message list.

5. Other Table Triggers

Other triggers exist that one may run into. For example, triggers when working with the wiring tables may occur. The previous sections should give you enough information to get you started with handling them. After working with the database, it will become easier to make an informed decision on how to handle a situation. Always be aware of the structure and intertable relationships when modifying the database.

Glossary

buildmb	Uses a set of SQL queries and formulas to calculate values for M and B of DescRec entries.
convertX	This will calculate the X coordinates for fields based on their recid in a crt entry.
convertY	This will calculate the Y coordinates for fields based on their recid in a crt entry.
dbconvert	Convert the Postgres database entries to the AccelNET runtime format.
dbmodify	Send the converted Postgres database entries into the currently running runtime system. This is only safe for a select set of tables.
dump_accelnet_db	Creates flatfiles of all Postgres tables for a database.
fetch	Retrieves database entries from the Postgres database as well as customizable SQL queries. The arguments vary for this command based on the current directory.
pg_dump	Used to get a raw dump file from Postgres for a database.
pgEdit	Experimental graphical database editor.
Query Files (.pqry)	These files are used to customize a fetch command.
update	Sends updated database entries, including new additions to the Postgres database. If a customized SQL query was used to retrieve the entries, the query file must remain in place.