SpecTcl Tutorial
7/3/12 Version

Connecting to Sorting Computers and LINUX basics

Currently we have six LINUX machines that will be available for offline sorting: auger, balmer, fourier, oppy, stern, and stark. These are all "headless" rack PC's in CW31 that can only be accessed remotely via puTTY or Private Shell. These can be accessed from the start menu on your PC. If they are not there, see the guys in PCSC. **PuTTY** first dumps you into a Configuration screen. You can configure sessions to different computers and Save them for later recall with the Load key (they will show up under the scrolled selection box under Saved Sessions). The three most important settings to check are that The Backspace key is set for Control-?(127) under the Terminal->Keyboard section, the Enable X11 Forwarding box is checked in the Connection->SSH->X11 section, and the Host Name of the machine you want to connect to is entered (e.g. balmer.phys.ksu.edu). These settings can be saved by clicking on the Session menu inside the puTTY window, then typing a name in the Saved Sessions box to match the name (e.g. balmer) in the Host Name box, then clicking on the Save button. **Private Shell** also begins in a "Log in – Server parameters" screen that lets you recall saved setups or create new ones. Here, you mainly want to check that the "Forward remote X11 connections to local display" box is checked in the Tunneling menu. For X windows to work correctly, the Xming X server must be running on your machine. This should have happened automatically when you logged in, as indicated by a black X in the lower right system tray of your status bar at the bottom of the screen. If it's not there, see PCSC.

To sort, you should log into one of the above machines as ##online, where ## is replaced by the initials of your group leader (lc, ib, vk, ct, mk, ar, or bd for now). All people in each group should log in using the same ##online account and password, set by the group. Once logged in, you are on a LINUX box, so Microsoft Windows commands no longer work. LINUX is case sensitive, so don't forget that. If you want to be able to open several windows without logging in again, do a

```
konsole &
```

This will run the konsole terminal program in the background, and clicking New Shell in the Session menu will give you another terminal session. You can switch between sessions with the tabs at the bottom of the window. Private Shell will let you open other windows with the Terminal button. (Note that the ampersand (&) at the end of any LINUX command means to run in the background and return control to the prompt. Otherwise, you can't do anything else in that window until the program you are running is finished.) To change directories, do a

```
cd ~/lcgroup/dray
```

for example. (The tilde (~) translates to the home directory of the user logged in, in this case lconline. Note that each group has a directory defined as ##group, where ## are the same group initials used in the ##online login name. All files under this ##group directory are located on the departmental networked SANS and are backed up regularly by PCSC. Anything above the ##group directories, say in a directory ~ibonline/mydir, is on the local LINUX machine and not backed up.

To create a directory, use the command mkdir. For example, to create the directory mydir under Itzik's SANS group directory, use

```
mkdir ~/ibgroup/mydir
```

To copy a file to mydir from another directory called olddir, use:

```
cd  ~/ibgroup/mydir
cp  ~/ibgroup/olddir/oldfile.txt  .
```

(That's a _space_ period at the end, important. The period means the current directory in LINUX.)

This will put the file oldfile.txt in mydir, using the same name. To copy a directory and all of its files to another directory, use the following:

```
cp –r ~/ibgroup/olddir/fildir ~/ibgroup/mydir
```

This will create the directory ~/ibgroup/mydir/fildir containing all of the files the ~/ibgroup/olddir/fildir contained.

A few other useful LINUX commands are:

List the contents of a directory:
ls (add -l to see more details, add *.xxx to see files of that form)

Move a file
mv location/filename newlocation/newfilename

Remove (i.e. delete) a file
rm location/filename.xxx

See the full documentation for a command:
info command (use 'Ctrl'-c to exit)

Get a man page (help documentation) for a command, sometimes shorter than info
man command

Search text files for a word

grep word filenam (use -i for non-case sensitive, add *.XXX to search files of that form)

See the contents of a file one page at a time
more name.xxx (this will type the file on screen; press the spacebar for the next page)

The above commands allow you to manipulate files with the command line. Many prefer to do so graphically, as in Windows Explorer. A good tool for this is the Web Browser/File Manager Konqueror. To run, type:

konqueror ~ &

on the command line. The tilde (~) will start konqueror looking at the ##online directory. You can replace it with any starting path that you wish, such as ~/lcgroup.

TCL (Command language) Files

You will notice that you have several *.tcl files in your sorting directory. These are Tcl/Tk command language files, analogous to the old *.com parameter files on the VAX. Tcl (pronounced "tickle") is an open source scripting language that is widely used. Tk is the graphical user interface toolkit for Tcl. Lots of online resources and published books can help you be as productive in this language as you wish, but for now we'll just focus on the SpecTcl specific parts. One important Tcl file is usually called setup.tcl. This is where your parameters and spectra are defined and other setup tasks performed. SpecTcl must have a parameter defined for every entity you want to histogram. A spectrum is then associated with that parameter. If you have a 2D spectrum, you must define two parameters, one for the x axis and one for the y. As an example, consider the following two lines from a setup.tcl file.

```
parameter adc8 108 12
spectrum adc8 1 adc8 12
```

The first line defines a parameter called adc8. Each parameter must be assigned a number, and this one is assigned number 108. This number will be used inside the sorting code to actually increment the parameter. This parameter is set up to hold numbers up to 12 bits in length, or 4096, i.e. a 4096 channel spectrum. The next line defines a spectrum that displays the histogram of the variable. The first adc8 is the name of the spectrum (it's fine to use the same name for both parameter and spectrum). The 1 means it's a 1D spectrum. The second adc8 is the parameter that the spectrum is histogramming, and the 12 means it, too, is 4096 channels. It is also possible to define parameters and spectra with real ranges, such as:

```
parameter rtof4 223
spectrum rtof4 1 rtof4 {{0. 3000. 1000}}
```

Here, parameter 223, called rtof4, will be histogrammed by a spectrum with a real range, and so it doesn't need a range value. The spectrum range definition is more complicated: It runs from channel 0. to channel 3000. and has 1000 bins. Note the double braces, required for the definition.

2D spectra can also be defined, for example:

```
parameter pipicox 240
parameter pipicoy 241
spectrum pipico 2 {pipicox pipicoy} {{0. 2000. 400} {0. 2000. 400}}
```

Note that I have to define a parameter for both the x and y variables. By comparing to the 1D definitions, this should be self-explanatory.
NOTE: the rules on names you can use are very lenient. There is no reason (other than typing) to use short parameter names or to have the same spectrum name as parameter name. For example, this would be perfectly acceptable:

```
parameter 2nd_recoil_time_of_flight 240
parameter 1st_recoil_time_of_flight 241
spectrum tof_coincidence 2 {2nd_recoil_time_of_flight 1st_recoil_time_of_flight) {{0.
2000. 400} {0. 2000. 400}}
```

It is also fine to use the same parameter for several spectra, as long as the parameter ranges are the same in each.

The spectra defined in setup.tcl are not automatically available for plotting. The command

```
sbind spectrumname
```

will assign a display slot in the display program (called Xamine) to the new spectrum named spectrumname. You can also define a series of spectra and then issue one

```
sbind –all
```

command at the end. This is typically what's done in setup.tcl.

*.tcl files can also be used to set other variables that can be used inside the sorting code. This is as an alternative to using constants in the sorting code itself and rebuilding the code each time a change is made. If only a few parameters need to be set, they can be incorporated into the setup.tcl file. If there are numerous parameters, it is often more convenient to create a separate parameter file and read it in before sorting. (How this is done will be described below).

C++ Tips

All of the sorting code itself is written in C++. In many ways, C++ is very similar to the C language, but with some extensions, the biggest being the concept of Object Oriented (OO) programming: classes, hierarchies, inheritance, etc. Fortunately, you don't need to understand much about OO to modify and even write successful sorting code. If you know only Fortran, here are a few tips (in no particular order) to remember about C++:

- No indentation is required, but it is often used to make code sections and iterative loops more readable.
- Comments can either be as in C, where /* starts a comment and */ ends it, no matter how many lines are in between, or the C++ specific //, meaning everything following on the same line until the carriage return is a comment.
- Each line of a C or C++ program (with some exceptions) must end in a semicolon, " ; ".
- Header files, basically chunks of code that get added to the source code and often have the extension .h, are included with an #include command. If the command is in the form #include "filename.h" , with the header name in quotations, the compiler (actually the "preprocessor") looks for the file in the local directory. If it is written #include <filename.h>, it looks in one of the standard include directories that the compiler knows about. (This can be changed in the Makefile).
- There is no implicit variable typing. All variables must be explicitly defined, using, for example, int for integer and float for real. Other modifiers can be used in front of these, such as constant or unsigned. The author of SpecTcl has defined shorter names for many of the combined types for use in the program, such as UInt_t for unsigned int.
- Arrays use square brackets [], not parentheses (), as in evarr[256]. Typically, the array index starts at 0, so that a 256 element array runs from 0 to 255. Two dimensional arrays use multiple brackets, as in rec[3][16].
- Output can be done easily by using the operators << and >> and names for standard error and standard output, cerr and cout. For example, the line cerr << "Too many hits on channel " << chan << endl; replaces the variable chan with its value and writes the line to standard error, which will appear on your konsole screen. The variable endl has been defined to mean end-of-line and carriage return.
- Variables can be incremented or decremented by 1 with the ++ or -- operators, as in count++; to increment the variable count.
- Braces { } are used to mark off sections of code that go together, such as subroutines, if blocks, or loops (for and while loops).
- Logical comparisons in if statements use ==, <, >, <=, >=, ! (for "not"), && (for "and"), and || (for "or"). One of the most common mistakes in C or C++ is to use a single = for equality comparisons. The statement if (a = 1) actually sets the variable a equal to 1 instead of giving a logical value if the two are equal (the proper form is if(a == 1)).
- Pointers are often a difficult concept to grasp. A pointer is a variable whose value is the address of another variable, not its value. So, assume we have an integer variable ival. The pointer to ival would be declared with int* pval; meaning that pval points to an integer. Then, pval can be assigned with pval=&ival; where the

& operator means "address of". If pval is incremented, with pval++, it means that it now points to the next address location. If instead the form *pval is used, as in (*pval)++, the content of the address pointed to by pval, ival, is incremented by one. Obviously, proper use of parentheses is important here. This is especially useful when passing variables as arguments to subroutines. By default, these variables show up in the subroutines as copies of the originals, so changes made in the subroutine to the copies are not made to the originals once the program returns from the subroutine. However, if pointers are passed as arguments, the subroutine can make changes to the original by using the *pval form, since the copy of the address still works as the address of the original variable.

- Instead of the do loop in Fortran, C and C++ use a for loop. The format is

```
for(int i = 0;i < imax;i++) {
.
.
}
```

The variable i is only defined for the duration of the loop, so it must be defined as an int and initialized. The loop starts with i = 0 and stops when i is greater than or equal to imax. i is incremented by 1 for each iteration. The braces aren't necessary if there is only a single statement in the if block. A similar loop is the while loop, which continues to execute the loop as long as the condition in the while statement is true. For example,

```
while (*pl != 0x0000C0FF) {            // Execute as long as the value at pointer
// location pl is not equal to Hexadecimal 0000C0FF.
  evarr[evCount] = *pl++;      // Copy the value of the variable pointed to by pl
// into array evarr, then increment pointer to point to next item in buffer.
  evCount++;                   // Increment event counter.
 }
```

- C and C++ don't have as many built in math operators as Fortran. In particular, there is no operator for taking a number to a power, such as ** or ^. If something just needs to be squared, it's easiest to write it out as a multiplaction (ivar * ivar). For larger powers, library routines can be used (pow(d,e), d to the power e).
- The OO idea of classes does directly affect how "subroutines" are defined in C++. A class is typically first declared in a header file with the class keyword, and all subroutines that are part of that class are declared within the braces { } of the class. For example,

```
class CLVEventDecoder : public CEventProcessor
{
void Resort(int arg1, int arg2);
.
}
```

defines the class CLVEventDecoder, which is based on the class CEventProcessor, and declares one of its routines Resort. Then, when the routines are actually defined, usually in the *.cpp file, they are referred to by the form: classname::routine, such as CLVEventDecoder::Resort(int arg1, int arg2). This means that the routine Resort is part of the class CLVEventDecoder.

- Each class has "private" data that can only be accessed by subroutines in the class, sort of like Fortran "common" in that it survives between subroutine calls. The data variables are defined in the class definition in the header file and typically initialized in a class subroutine called the constructor (just a subroutine with the same name as the class). The initialization can be done with a (0), such as

```
CLVEventDecoder::CLVEventDecoder() :
  m_nEventCount(0),
  m_nGoodCount(0)
{
}
```

  where the private data variables m_nEventCount and m_nGoodCount are both initialized to 0.

Remember, C++ is a very powerful language, just like Fortran, and it will take a lot of work to master it. However, these few tips will hopefully answer many of the questions that come to mind when looking at C++ source code for the first time.

SpecTcl Sorting Code Logic

SpecTcl sorting code must conform to the structure that SpecTcl requires. The first step in understanding that structure is to look at the sorting class subroutines that have special meaning. If the class name is CLVEventDecoder, for example, these routines are CLVEventDecoder::OnAttach, CLVEventDecoder::OnBegin, CLVEventDecoder::OnEnd, and CLVEventDecoder::operator(). Commands included in the OnAttach routine are run when SpecTcl starts up for the first time. OnBegin commands are run at the beginning of each run, assuming that the data buffer has a begin event. OnEnd is after an end run event. These should be relatively self-explanatory, but the operator() routine is the most non-intuitive from a Fortran point of view. C++ allows something called operator overloading, which is basically re-defining a standard operator to mean something else. In this case, the parentheses operator (), when used with the CLVEventDecoder class, is defined to carry out the commands in the operator() code (between the { and }). This use is actually something you will never see, as it is buried deep in the support code for SpecTcl. However, you only need to know that what happens in the operator() routine is the code that gets executed for each event. This is where the bulk of the sorting actually takes place. Other subroutines can be defined and called by the user from the basic SpecTcl routines, usually operator().

SpecTcl parameters are histogrammed in two ways, depending on if the parameter is to be incremented only once per event or multiple times per event. The first case is much easier, only requiring a line such as:

```
rEvent[208] = xr2;
```

which increments parameter 208 at position xr2 by 1. (Remember, the parameter is what is processed; the spectra are histograms of the parameter). This is different from other

systems like ROOT where a histogram increment is explicitly done. It also means that there is no simple mechanism to do multiple increments on a parameter in a single event, such as incrementing a time-of-flight spectrum parameter with all time hits. This requires manipulating the spectrum itself and involves a lot of code overhead. It looks daunting, but the basic format can be used for all multiple-increment spectra. Let's consider a multiple TOF spectrum. The first step is to make the spectrum part of the private class data with the line:

```
CSpectrum* m_prtofall;
```

in the class definition in the header file. CSpectrum* means that m_prtofall is a pointer to the class CSpectrum. The pointer is initialized to 0 in the event decoder constructor just like any other variable, with m_prtofall(0). Next, the actual spectrum has to be found for m_prtofall to point to.

```
string strrtofall = "rtofall";      // Define a string variable to hold the spectrumname
m_prtofall = pHistogrammer->FindSpectrum(strrtofall);      // Use the FindSpectrum
// subroutine in the class pointed to by pHistogrammer to find the spectrum named
// rtofall.  If found, m_prtofall will now point to the spectrum.
if(m_prtofall){                    // Make sure it found the spectrum, i.e. m_prtofall isn't 0.
 if(m_prtofall->getSpectrumType() != ke1D) {        // Make sure  it's a 1D spectrum.
  cerr << "Found "<<strrtofall<< " but it's not 1-D\n";          // If not print error and
  m_prtofall = (CSpectrum*)kpNULL;                              // reset pointer to 0 (Null).
 }
}
```

Finally, the spectrum is incremented as follows:

```
for(int i = 0;i<nrec;i++) {                // Here, loop through several hits in this event
  if(m_prtofall) {        // rtofall spectrum (make sure it has been properly found)
   newIndex = (UInt_t)(m_prtofall->ParameterToAxis(0,rtof[i]));    // Since this spectrum
// was defined with a real axis, an axis transformation is necessary, using the value rtof[i].
   if(newIndex < m_prtofall->Dimension(0)){          // Check to make sure the channel
// isn't out of range by checking against the spectrum's dimension.
    newValue = (*m_prtofall)[&newIndex] + 1;        // Increment the value currently
// at that channel.  Note that the definition of the spectrum requires the address of the
// index (channel), as &newIndex.
    m_prtofall->set(&newIndex,newValue);            // Actually set that channel's value.
   }
  }
 }
```

The same technique works for 2D spectra with minor changes to account for the second dimension. I have created functions called MultiPlot1d and MultiPlot2d to do these multiple increments. Note, however, that this method of multi-incrementing essentially bypasses some of the benefits of SpecTcl, such as being able to use real numbers

(including negative numbers) on the axes of spectra, and having access to the built-in gating commands. Other methods are available to do this within the SpecTcl context. See me for further information.

Often, variables that are seldom changed, such as physical constants or the size of a channel plate, are initialized in a header file with the word Constants somewhere in the name, such as CLVConstants.h or C1290Constants.h. Should changes be necessary, it's easier to locate the variables when collected in one place, change them, then rebuild the code. It is also possible to initialize some of these variables in a Tcl file so that the sorting code doesn't need to be rebuilt each time a change is made to a "constant". Please see me for information on how to do this, as it is beyond the scope of this description.

Building and Running the Sorting Code

Now, on to compiling, linking, and running the sorting code. In what follows, I'm assuming your current directory is the same as that where your sorting code resides, for example, ~/ibgroup/Xlasersort/SpecTcl. Once inside a sorting directory, you make sure that the executable file is up-to-date via the commands (in one of your regular Linux terminal windows, such as a konsole window)

```
make clean
make
```

The first command deletes all of the old compiled and linked files and is not strictly necessary if the make dependencies have been properly configured. However, it's safest to use it initially until you become more comfortable with the procedure. The second command recompiles and links the code. The result is that you will have a file called SpecTcl that is the executable used to run the program. By the way, a nice feature of the shell used on our LINUX box is command completion. If you type enough of a command or filename to clearly identify it from other possibilities, hitting the Tab key will complete the command/name for you. The rules used by the make command are found in a file called Makefile in the directory you are in. This is a very powerful file that makes compiling and linking up-to-date code much simpler, but learning how to modify the file will take some time. Usually, you shouldn't need to touch it.

To run the sorting code for the first time, from your sorting directory, type:

```
./SpecTcl < setup.tcl
```

(Note, for some groups, a parameter file needs to be read in before setup.tcl)

The ./ means look for the SpecTcl program in the current directory. The < is LINUX redirection, meaning to read in the file that follows. You should see four X objects pop up on your screen (some of them may be hidden, so look below on your taskbar in the X section to bring them forward if necessary). These are the gui window, the tkcon

window, the Xamine window, and the SpecTcl command buttons window. The Xamine window is where you will look at and manipulate spectra, the tkcon window is where all commands will be typed, the gui window allows you to get spectrum and parameter information and execute some commands via a gui menu, and the SpecTcl command buttons are for frequently used commands. The most common commands to execute are as follows:

- If you have done some of your variable definition in a Tcl parameter file called, for example, rerun1000.tcl, after modifying a parameter in that file, in the tkcon window type

  source rerun1000.tcl

- You need to attach an event file to let SpecTcl know which file to sort. This is done, naturally, with the attach command. If you are sorting data that was not taken with the data acquisition system, nscldaq, you will have to specify the size of the buffer, since it's different from the default. A command that will attach a Labview data file called datfil, for example, is

  attach -size 204816 -file /common/lcgroup/data/datfil

  where 204816 is the size of the buffer in bytes. To combine several data files into one sort, the following will work:

  attach -size 204816 -pipe cat /common/lcgroup/data/datfil1
  /common/lcgroup/data/datfil2
  /common/lcgroup/data/datfil3

  This takes advantage of a "pipe" in Linux that hooks up the output of one command, cat in this case, to the input of another, here the attach command. If desired, all of this can be put in a separate command file and sourced in from the tkcon window or even placed at the end of the setup.tcl file.

- To Start/Stop sorting, click on the Start/Stop Analysis button on the SpecTcl command window. Spectra can be cleared with one of those buttons, and 1D and 2D spectra can be exported to a text file for reading in to Origin.
- To look at spectra in the Xamine window, you first have to define a set of display panels and choose which spectra to display. Predefined sets of spectra may already be defined These can be accessed by the Window->Read Configuration buttons. The various window sets are displayed under the Files heading. Select one and click OK. If you want to define a new window set, click on the Geometry button at the lower right of the Xamine screen. This allows you to configure the display for up to a 10x10 matrix of spectra. After selecting your display geometry, you have to select specific spectra to go in each slot. The Display button allows you to select a single spectrum for a single slot. It produces a scrolled list of all the spectrum names you have defined. Selecting one with the mouse (or typing the name in the text box) and clicking OK will display that

spectrum in the display slot. The Display+ button will automatically step through each available slot as you select a spectrum and hit the Apply button. Use the OK button for the last slot you want to fill. If you want to save this particular display configuration, the Window->Write Configuration buttons will let you give it a name. It will automatically be given a .win extension.

- A given spectrum can be chosen to fill the display by double clicking on it. Double clicking again will reduce it to its former size. (The Zoom button does the same thing.) Part of a spectrum can be expanded by using the Expand button. A window will pop up to let you enter the coordinates at each end of the area you want to expand, or you can click on them with the mouse. Note that sometimes this window pops up behind the Xamine window and can be hard to see. You can look on the taskbar at the bottom of your PC in the X # Xming group to find the Expand window and click on its name to bring it to the front if necessary. The UnExpand button removes the expansion.
- To calculate the area of a peak in a spectrum, first define a Summing Region with the Xamine button of the same name, then click the Integrate button. These are simple integrations, i.e. a sum of the counts in each channel between the summing region boundaries. The centroid and FWHM are also calculated. For background subtraction or fitting, the data must be exported to Origin. Summing regions can be removed by clicking on the Graph_objects pull down menu at the top of the Xamine screen and selecting Delete. You'll get a list of graphical objects defined for that spectrum, including summing regions, and these can be selected to delete.
- To overlay one spectrum with another, click on the Spectra pull down menu at the top of Xamine and then click on Superimpose. To remove an overlay, click on UnSuperimpose.
- The Log button in the middle of the bottom part of the Xamine screen toggles back and forth between a log scale on the Y axis for 1D spectra or the Z axis for 2D spectra.
- The Map button next to the Log button toggles back and forth between displaying the channels as integers, 0 to the maximum channel in that dimension, or as the spectrum was defined, from the defined minimum to maximum channel numbers. If the spectrum was defined as a number of bits in size, these two will be the same.
- The Marker button puts a dot on the spectrum. The Cut button sets a 1D gate, Band sets a 2D gate. Read the more extensive online SpecTcl documentation for how to use gate commands.
- Other options can be accessed from the menu buttons at the top of the Xamine screen, such as setting Spectra properties like Autoscale. You can experiment with these.
- To print a spectrum, click on the File menu and select Print... Usually, I select Landscape, Print Selected Spectrum, and Specify size in Spectrum Options. Then, in the Spectrum Options tab, I set the size to be 9" wide by 6" high. To print all of the spectra on a multi-spectra display, select Print All Spectra and choose the number of Rows and Columns to match the display. Output options can be either To File or To Printer. If Printer, the Print Command has to have the right printer name after the lpr –P. Available printers and their names are:

acc4600	HP4600 color printer in lab, single side
acc4600d	" ", duplex
accebis	EBIS b&w printer
acchplj8150	HP8150 b&w printer in lab, single side
acchplj8150d	" ", duplex
co349000	HP9000 b&w printer in CW34, single side
co349000d	" ", duplex
c034c4600	HP4600 color printer in CW34, single side
c034c4600d	" ", duplex
c034dell5310	Dell b&w printer in CW34, single side
c034dell5310d	" ", duplex
c305lj4000	HPLJ4000 b&w printer in CW305, single side only

- Spectra can be saved and then read back in with the swrite and sread commands. The format of these commands can be found in the more complete online SpecTcl documentation. These are best for saving and reading a single spectrum. Although several spectra can be written to a single file with swrite, sread will only read the first spectrum in that file unless it is called from within a more complex Tcl procedure. Procedures have been written to write all defined spectra to an ascii file and to read those spectra back in later, replacing the contents of existing spectra. Typing wrtall in the tkcon window will pop up a graphical file dialog that allows you to specify the name and location of a file to write the spectra to. The result will be an ascii file that contains all defined spectra. To read them back in, type rdall, select the file from the pop up dialog, and wait for the procedure to finish. (This will take a few seconds depending on how many spectra you have and how large they are.) Remember that any existing spectra will be overwritten.
- To make sure the spectra are updated during the sort, choose Options and select Update Rate. Set the slide bar to 10 seconds and click Apply to All.
- There are two options for taking projections of 2D spectra. The first is a command that we have written in house in TCL code. The format is:

  Xproject sourcename ylow yhigh targetname
  Yproject sourcename xlow xhigh targetname

  The sourcename is an existing 2D spectrum and targetname is an existing 1D spectrum, which you must define in your setup.tcl file. For example, the lines

  parameter proj256 1256 8
  spectrum proj256 1 proj256 8

  define a 256 channel 1D spectrum to histogram parameter 1256. The low and high values are the channels limits to include in the projection sum. There are no default values, so these limits must be included. The spectra created with this command are static pictures (snapshots) of the source spectra at the time the

command is run. The projection spectra do not accumulate new counts if the sort is continued.

The newest version of SpecTcl as of this writing (SpecTcl 3.3) has a built in projection command. Its format is:

project [-[no]snapshot] sourcespec newspec x|y [contourname]

where you choose "x" or "y" depending on which direction you wish to project. The snapshot option works like our homemade X or Yproject commands. -nosnapshot creates a spectrum that is incremented when the sort is continued, or even cleared and restarted. Here, sourcespec is an existing 2D spectrum and newspec is the name of a new, nonexistent 1D spectrum. Instead of manual channel limits, an optional contourname can be given to limit the number of channels included in the projection. A contour can be set by clicking on a 2D spectrum, clicking the Contour button, and then clicking a series of points on the spectrum to form a contour, closed by the OK button. You have the option of giving the contour a name or using the default name provided. If you forget the name of a contour, you can click on the Graph_objects pull down menu at the top of the Xamine screen and select Copy Object, which will give a list of graphical objects defined for that spectrum. If a contourname is omitted, all channels along one axis of the spectrum will be summed for the projection. Before displaying the newly created projection spectrum, you have to issue the command sbind –all in the tkcon window to let Xamine know that the spectrum exists.

Help and Additional Capabilities

Aside from this document, help for SpecTcl can be found at the following URL:

http://docs.nscl.msu.edu/daq/spectcl/

This was written by SpecTcl's author, Ron Fox, at the Cyclotron Lab at Michigan State University. All of the built-in SpecTcl commands listed in this document are explained there in greater detail.

Clearly, Xamine is not perfect as a spectrum display program. In particular, the spectra aren't publication quality. However, the author of the code is continually upgrading it and seems responsive to suggestions. It is also a very easy thing to export spectra and read them into Origin where they can be prepared for publication. Also, event files can be exported in a format that can be read in by the CERN program ROOT (a successor to PAW). ROOT is a powerful package that has numerous built-in analysis tools as well as superior graphics.

As experiments are becoming increasingly complex, event files are growing larger and the associated sorting time is increasing. For some experiments, many of the events fail to satisfy some basic conditions and can be ignored. In that case, further sorting on a

given run can be greatly speeded up by writing out a filtered data set and then performing subsequent sorting on the filtered data. Typically, the filtered data file is much smaller than the original and can be much quicker to sort.

For information on these additional capabilities or if you have further questions, please contact Kevin Carnes, kdc@phys.ksu.edu.