Tandem SpecTcl

W. A. Peters
National Superconducting Cyclotron Laboratory

February 13, 2006

1 Getting Started

The tandem version of SpecTcl for the Sweeper-MoNA setup is a complex and large set of files compiled to form a complex and powerful analysis tool. It unpacks all the raw channels and converts many of them to calibrated parameters and even some to very useful physics parameters. Please familiarize yourself with the *Mona SpecTcl Guide*[6] and the online *SpecTcl User's Guide*[1], before proceeding. Concepts in those documents will not be reviewed.

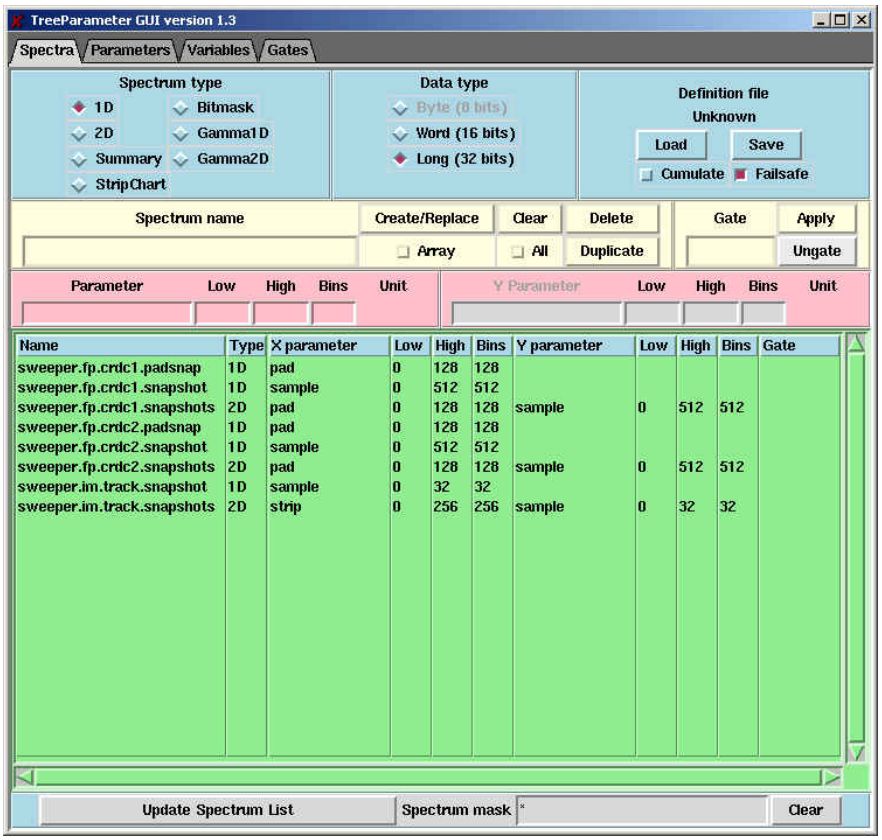
One of the most noticeable differences is that an extra control window is used with the tandem SpecTcl. This `Tree Parameter GUI` has four tabbed pages (`Spectra, Parameters, Variables, Gates`). These will be discussed in detail in section 2.3. The other main visual difference is the SpecTcl Control Window has two graphics in the center and a few extra buttons to track forward and do inverse tracking to the target. These extra features will be discussed in section 4.

1.1 Tandem_spectcl.tcl

In the Tandem version, this file replaces the `MoNA_spectcl_run.tcl` file and also creates a couple tandem parameters and set a couple tandem variables. Many of the MoNA variables are set by sourcing `MoNA_hardware_run.tcl`. One of the tandem variables it sets is the extra energy to add to the fragment energy for e-loss through half of the reaction target (`target.eloss`) in units of MeV/u. The `MoNA_Zpos` variable is set in this file to the number of centimeters from the target to the center of bar MoNA bar A8.

1.2 Loading a Settings File

- The front page of the *Tree Parameter GUI* looks like this:



The button *Load* in the upper left corner can be used to load a file a definition file containing all the spectra, gates, and variable settings and values. Using the *Cumulate* option will allow you to keep the current settings and adding the differences from the new input settings file. The current settings for all spectra, parameters, variables, and gates (that can be edited using the directions in the next section) can be save by using the *Save* button. These files should be stored in the `/settings` directory.

This first page of the *Tree Parameter GUI* also is where the spectra can be created (for `sweeper` and `user` tree parameters only).

2 Tree Structure

The Sweeper, and some MoNA, variables and parameters are defined in a tree structure. The two base tree labels are `sweeper` and `user`.

2.1 Sweeper Tree

This tree contains all the sweeper parameters and variables written by D. Bazin, including:

```
fp (CRDC's, thin and thick scintillators, IC)
im (beam-line tracking)
```

```
tof (rf, xfp, potscint timing)
trigger (tigger channels)
```

These branches hold parameters like CRDC/beam-line tracking positions and angles, Ion Chamber energies, thin and thick scintillator energies and various timing channels.

2.2 User Tree

This tree contains calibrated parameters created by W.A. Peters. Many of the Sweeper parameters cannot be calibrated without major changes to the code, so this tree was made to input many raw Sweeper parameters and then calibrate them with user defined variables. Branches of this tree include:

```
tof (rf, xfp, potscint, and thin scintillator timing, also fragment velocity)
e (thin and thick e-loss)
```

These branches hold parameters that can be calibrated and manipulated within the `CUser` tree code.

2.3 Using the Tree Parameter GUI

2.3.1 Making Spectra The following directions apply only to `sweeper` and `user` tree branches. For MoNA spectra, use the TkCon window or the pre-set buttons on the SpecTcl Control window[6]. MoNA spectra will indeed be displayed in the spectra section and can be edited once created. D. Bazin has written some excellent documentation about using the Tree Parameters[2].

1. In the upper left hand corner of the window, select the type of spectrum you wish to create. For information about the different types see Ref[1].
2. Next, select *X Parameter* drag-down menu and move cursor to find the tree parameter you want.
3. The default parameter limits will be displayed to the right of the name.
4. Edit the parameter limits `Low, High, Bins` as desired.
5. Select *Create/Replace* and your spectrum will be displayed in the columns below.
6. Double-click any spectrum to edit any of the limit values and then repeat previous step.
7. Apply any gate that is made by selecting the Gate drag-down menu and then *Apply*.
8. Save time by saving definition file using *Save* button.

2.3.2 Changing Parameters On the second page you can change the default settings for any parameter. These are the same limit values that are displayed when a parameter is selected for a spectrum, as described above.

1. First click the designation button for an empty row. It will turn red.
2. Select a parameter from the *Parameter* drag-down menu.
3. The default values will be displayed.

4. Edit the values for `High, Low, Bins, Unit`.
5. Select *Set* to make the changes.
6. Select *Load* to revert back to previously saved values.
7. Select *Change Spectra* to apply changes to all spectra containing that parameter.

2.3.3 Changing Variables Changing tree variables on the third page of the Tree Parameter control window is down analogous to changing parameter values as in the previous section.

1. First click the designation button for an empty row. It will turn red.
2. Select a variable from the *Variable* drag-down menu.
3. The default values will be displayed.
4. Edit the values for `Value, Unit`.
5. Select *Set* to make the changes.
6. Select *Load* to revert back to previously saved values.

2.3.4 Changing Gates The forth page in the Tree Parameter GUI deals with gates. Gates can be made by using in-line commands in the TkCon window[6] or by using *Contour* or *Cut* buttons in Xamine. If any gates have been made you can double-click the name of any gate and view its dependencies and type. You can then edit the values or delete the gate. Compound gates can be made by using the in-line command, or selecting the `And` type.

3 SpecTcl Code Procedures

3.1 Raw Sweeper Parameters

The Sweeper code does not calibrate any timing parameters but does indeed have many other parameters that are very useful. The most important code for the Sweeper parameters deals with the four tracking detectors. The two beam-line trackers (PPAC's or CRDC's) and the focal plane CRDC's are read in through an XLM module and a large chunk of data is taken for each valid hit. Gas tracking detectors have many parallel wires that collect the ionizing charge from gas as a particle passes through. There is also a time channel to determine the drift time (usually the y direction) of the ionized electrons.

Another gas detector the Sweeper code reads, is the Ion Chamber. This works in a similar fashion as the CRDC's but is not used to determine the position. It has 16 charge collecting pads that are used to get a precise measurement of energy-loss through the gas.

The thin and thick plastic scintillators at the end of the focal plane are used for timing and e-loss measurements. These two, each have four PMT's and are recorded by eight ADC and TDC channels. The cyclotron RF and the A1900 extended focal plane timing detector (xfp), along with the pot scintillator finish off the sweeper timing channels.

Each time is common started by the `sweeper.trigger`, which is a logic-delayed signal from the upper left PMT on the thin scintillator. So, the `thin_ul_time` raw parameter has the same raw spectra

as the raw trigger parameter. Every timing channel immediately subtracts this `sweeper.trigger` raw parameter from itself in the code, to get ride of the logic unit's time jitter when it delays the PMT signal./footnoteSubtracting un-calibrated raw time parameters is not wise, but these timing channels are all on the same TDC and have very nearly the same slope to nanoseconds. The subtraction looks like this in the code:

```
thin.tul = thin.time_ul - sweeper.trigger (similar for all four PMT's)
pot = pot - sweeper.trigger
rf = rf - sweeper.trigger
xfp = xfp - sweeper.trigger
```

The raw parameters for RF, xfp, and pot times are lost, while for those for the eight thin/thick times are not because a new raw parameter is used for the subtracted values.

3.2 Tree Calibrations

The Sweeper code does not calibrate the timing channels, but is does have a calibrated energy-loss parameter for the Ion Chamber (IC) and calibrated position and tracking parameters. The User code calibrates the timing channels and also the thin/thick e-loss (done within the `CUser.cpp` file).

3.2.1 User Potscint, RF, XFP These are calibrated timing parameters that use the respective slope/offset values to calibrate the corresponding raw Sweeper timing parameter. Potscint, for example:

```
user.tof.potscint.cal = sweeper.tof.pot * user.tof.potscint.slope +
potscint.offset
```

Note, that the slope for each raw Sweeper timing channel might not be the same. Set the offset for the potscint from a known beam through the Sweeper magnet to the thin plastic. Be sure to calibrate and save the definition file before trying to use the third level parameters described below.

3.2.2 User Thin Time The timing channels for the four PMT's of the thin scintillator are calibrated and then averaged:

```
user.tof.thin_ul.cal  = sweeper.tof.thin.tul * user.tof.thin_ul.slope +
user.tof.thin_ul.offset (similar for all four PMT's)

user.tof.thin_time.cal = (ul.cal + ur.cal + dl.cal + dr.cal / 4) *
thin_time.slope + thin_time.offset
```

The offset should be set to place thin_ul at zero, and the other three thin times at, or near, 1 ns. The offset for the average, thin_time, should be set to place the peak at about 0.5 ns, it's slope should be negative (so it's ready to be subtracted from the potscint time).

3.2.3 User Thin/Thick e-loss These parameters are analogous to the timing channels. Here each thin/thick Sweeper raw ADC channel is calibrated and then the four are averaged. In addition, the two averaged energy-loss parameters are added together to form `user.e.total.cal`.

3.2.4 Sweeper Positions and Angles Variables can be set to calibrate the gain and pedestal values for each CRDC pad. The x position is calibrated by taking the weighted average of all the pads above threshold and then, using the slope (2.54 mm per pad), is converted to millimeters. Using a mask to calibrate the y direction (from the drift time), these positions can have a precision of better than one millimeter.

The angles are then calculated from these positions and the distance between them (also a variable that can be set). Furthermore, you can set the `track.zfp` variable and and get the calculated x,y positions at this z point in front or behind CRDC1. These two extra x,y parameters are called `track.xfp` and `track.yfp`. These two tracking parameters will become important later.

3.2.5 Sweeper Ion Chamber Each pad for the IC has it's own slope and offset variables and then the overall e-loss (`fp.ic.de`) parameter has a slope and offset.

3.3 Third-Level Parameters

Many of the calibrated parameters are still not at the level of being useful for a true physical analysis of the reaction of interest. The third-level parameters are used to calculate some useful ones.

All the other `user.tof` parameters are differences of two calibrated times. For example:

```
user.tof.rf_pot.cal = user.tof.rf.cal - user.tof.potscint.cal *
rf_pot.slope + rf_pot.offset
```

The respective slope/offsets should not need to be other than 1 and 0 since the components are already calibrated.

3.3.1 User Pot_Thin Time Having the potscint and thin_time calibrated parameters don't tell you the time-of-flight of the fragments until you subtract the two. The parameter for `potscint - thin\_time`, as described above for the difference, is constructed similarly here, but for the feature to adjust the time based on the tracking angle and position:

```
user.tof.pot_thin.cal = user.tof.potscint.cal - user.tof.thin_time.cal +
sweeper.track.xfp * sweeper.track.potthinxfp +
sweeper.track.afp * sweeper.track.potthinafp
```

To get the unadjusted tof, just set these adjustment variables to 0. The first line of this calibration is saved as the `frag_time` in the code and used for the fragment's velocity. This adjustment is useful for separating isotopes of the same element by rotating the time about the dispersive tracking position and angle.

3.3.2 User Fragment Velocity Here the `user.tof.pot_thin.cal` parameter is used with out the adjustments described above, along with `user.tof.vfrag.offset` in centimeters, to get the fragment velocity in cm/ns.[1] The `user.tof.vdiff.cal` parameter is the difference (in cm/ns) between the neutron velocity (calculated from hit 1) and the fragment velocity (it's slope and offset should be 1 and zero, respectively).

[1] You should follow the directions form section 2.3.3 and change the units for this offset to cm.

3.4 Tracked Parameters

By using an inverse map, the SpecTcl code calculates not only the position and angle of the fragment at the target, but also to relative energy to the magnet's center track. The most important parameters calculated by this method are:

```
 sweeper.track.ata,bta,xta,yta, and
target.de, the same as sweeper.track.dta
```

These are used to further calculate the energy of the fragment at the midpoint of the target. The `CHitParams.cpp` file is where these complex calculations are coded:

```
 target.e = frag.mass  * target.de * target.eo +
target.eoffset

  // solve for momentums from N_mass and N_vel and
  // m_fFragmass and Frag_vel (Pn and Pf)
       double Namu = Nmass/amu;
       double Famu = m_fFragmass; //Input Fragment in units of amu.
       double N_beta =  pow (( Nvel[first_count]/Vc),2.0) ;
       double N_gamma = sqrt(1.0/(1.0 - N_beta));

       // amu in units of MeV/amu
       // ResultTE is MeV/u
       // Frag_KE in units of MeV/u
       // m_fFragmass in units of amu

       double Frag_betagamma_o = ( m_fFragBrho / 3.107 * m_fFragQ / Famu);
    double   Frag_gamma_o =  sqrt(Frag_betagamma_o * Frag_betagamma_o + 1.0);
       double   Frag_KE_o =  amu * (Frag_gamma_o - 1.0); //in MeV/u

       // set energy from delta and central_energy
       // in MeV/u add m_feloss (target.eloss) to target_e
       double Frag_KE = ( ((double)rEvent[m_nParamDT] + 1.0) * \
        Frag_KE_o) + m_fTargeteloss ;

       rEvent[m_nResultTE] = Frag_KE; // in MeV/u
```

Where `target.eo` is the magnet's central track energy (in MeV/u) set from the *set central eo* button on the Tandem SpecTcl Control window after the mass and central B rho values are set. The target.eoffset variables is the estimated energy the fragment losses half way through the target (in MeV/u) and it is added the calculation to get the energy at the midpoint.

3.5 MoNA Tandem Parameters

The Tandem MoNA code calculates some useful parameters as well. The `Tandem_vdiff` hit parameters are miss-named and are actually just the velocity for that MoNA hit (in cm/ns) and not a difference at all. It is calculated from the ToF_hit parameters and the `MoNA_Zpos` variable.

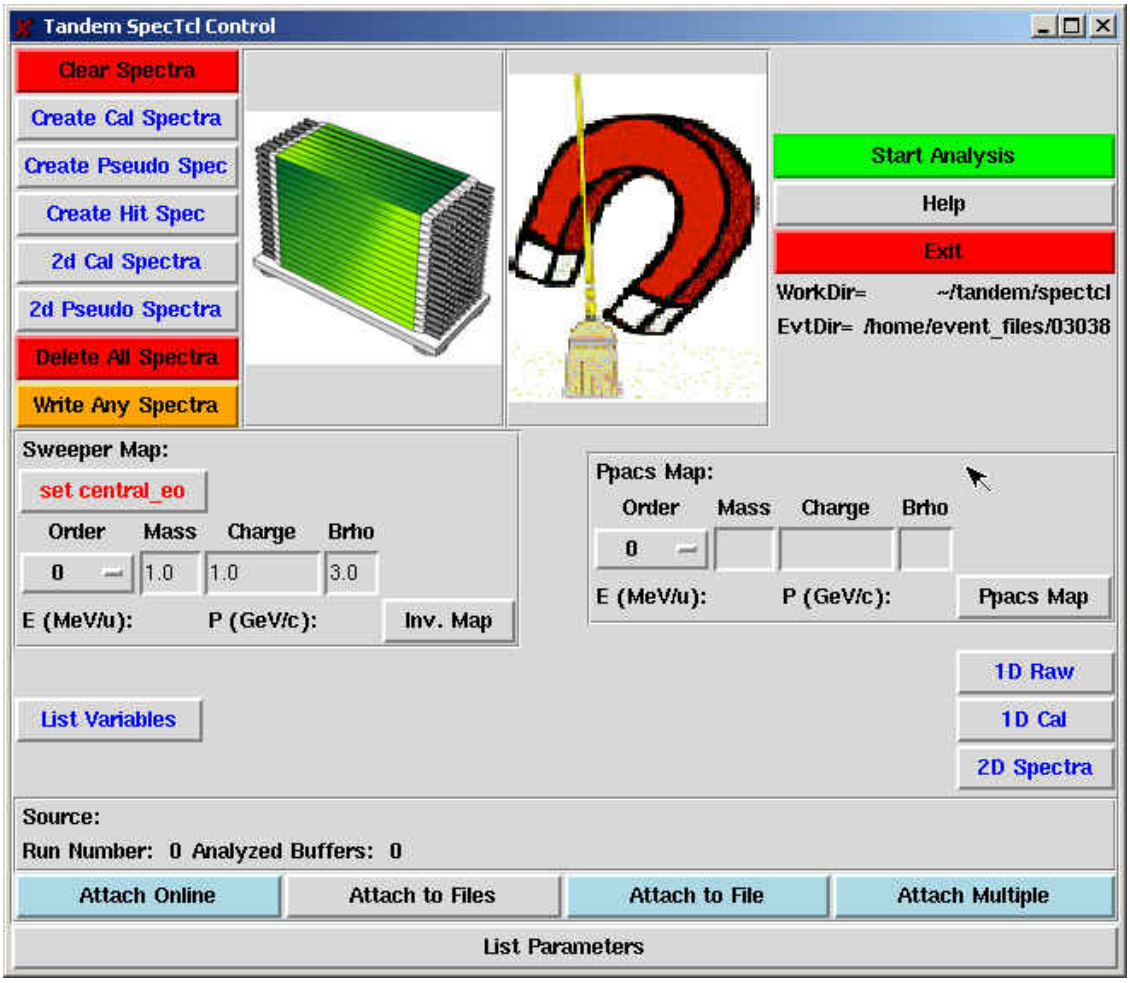
The KE_hit parameters are calculated form the velocity and the mass of a neutron. A good practice is to gate this parameter on the neutron peak from the ToF_hit parameter.

3.5.1 Decay Energy The angle of the fragment between the fragment and the neutron is calculated from the fragment tracked angles and the neutron angle. This goes into the Theta parameter. The energy of the fragment at the target (calculated using an inverse map) along with the Theta parameter and the neutron energy is used to calculate the invariant mass of the decay state by:

```
double Both_Sum = Frag_x * Xarray[first_count] + \
    Frag_y * Yarray[first_count] + \
    Frag_z * Zarray[first_count]; // in cm

   double Frag_R2 = (pow (Frag_z,2.0) + pow (Frag_y,2.0) + \
   pow (Frag_x,2.0)); //r-vector squared

   double Frag_distance = (pow (Frag_R2,0.5));  // in cm
   double Both_Product = Distance[first_count] * Frag_distance ;

   // solve for cos(theta) = AxBx + AyBy + AzBz / A*B
   double cos_theta = (Both_Sum / Both_Product);
   rEvent[m_nResultThe] =( acos(cos_theta) * 180.0/PI);

   //from KE(MeV/u) to gamma(unitless), with  c=1
   double Frag_gamma = 1.00 + (Frag_KE / amu);
   double Frag_beta = ( 1.0 - (1.0/(Frag_gamma * Frag_gamma)));

   rEvent[m_nResultKV] = (sqrt(N_beta) - sqrt(Frag_beta)) * Vc ;

   double Pneutron = Namu * sqrt(N_beta) * N_gamma;
   double Pfrag = Famu *  sqrt(Frag_beta) * Frag_gamma;

   //Calculate M0^2 + PMo^2 =  m1^2 + m2^2 +
   // 2( En*Ef - Pn*Pf*cos(theta))
   //first convert KE into total E for N and Frag

   double Eneutron = N_gamma * Namu;  // Total N E
   double Efrag = Frag_gamma * Famu ; // Total frag E
   double Etotal = Namu*Namu + Famu*Famu +  \
 2.0000*(Efrag*Eneutron - (Pneutron*Pfrag*cos_theta));

   double Qamu = sqrt(Etotal) - Namu - Famu; // in amu/c^2

   rEvent[m_nResultEdecay] = Qamu*amu; // MeV/c^2
```

4 Analysis Suggestions

The following sections are written to help with some of the standard analysis procedures. The Control window for Tandem SpecTcl is a little different than for the MoNA version:



4.1 Tandem Control Buttons

For the most part, the buttons on the Tandem SpecTcl Control window are the same as for the MoNA SpecTcl[6] but for a few exceptions.

4.1.1 Customized Buttons The `tandem/spectcl/tcl` directory contains a few files that are sourced by the gray buttons with the blue lettering. The *1D Raw* button sources the `tcl/SpecGen.tcl` file.The *1D Cal* button sources the `tcl/CalcParameter1DSpectra.tcl` file.The *2D Spectra* button sources the `tcl/CalcParameter2DSpectra.tcl` file. These three files can be edited to include any custom commands or spectra you choose.

4.2 Sweeper Inverse Tracking

Inverse tracking can only be completed after the CRDC's have been calibrated for position from mask runs. The *Inv. map* button is used in conjunction with the `Order, Mass, Charge, Brho` input frames to

calculate the energy of the fragment at target from information from the focal plane CRDC detectors. These inputs should correspond to the mass and charge of the fragment you wish to track through the Sweeper magnet. The `Order` should be set to 3 (or higher if the inverse map was made to a higher order). The Brho input should correspond to the Brho value setting of the Sweeper magnet before the run. Check the experiment's logbook for this number. This number is also needed to make the inverse map, so you can check with person who calculated the inverse map as well.

Once the input frames are filled with the appropriate values press the *set central_eo* button to set the `target.eo` variable. Next, the `sweeper.fp.track.map.maxparameters` variable should be set to 4 if you wish to use the old inverse maps, or to 5 for the new inverse maps that take into account the x position at the target from the forward tracking map. Now an appropriate inverse map must be made (.inv extension) and sourced with the *Inv. map* button. A good check to see if it working is to source a collimator run with a known beam and known energy. The positions at the target should be centered near zero, and the energy should match.

4.3 Beamline Forward Tracking

Forward tracking can only be completed after the tracking detectors have been calibrated for position from mask runs. The *Ppacs map*[2] button is used in conjunction with the `Order, Mass, Charge, Brho` input frames to forward track the position and angle of the fragment at target from the information from the beam-line tracking detectors. These inputs should correspond to the mass and charge of the fragment you wish to track through the triplet magnet. The `Order` should be set to 3 (or higher if the forward map was made to a higher order). The Brho input should correspond to the Brho value setting of the Sweeper magnet that was used to match the triplet before the run. Check the experiment's logbook for this number.

Once the input frames are filled with the appropriate values, make sure the `sweeper.im.track.map.maxparameters` variable is set to 4. Now an appropriate forward map must be made (.map extension) and sourced with the *Ppacs map* button. A good check to see if it is working is to source a collimator run with a known beam and known energy. The positions at the target should be centered near zero.

4.4 MoNA Timing

Make sure to have pot scintillator common stop signal. Set `tmean_offset` to move gamma peak to time light takes to travel `MoNA_Zpos` centimeters. Check *List Variables* button to display current values.

4.5 Fragment Separation

For detailed directions on this subject see Ref[3].

References

[1] R. Fox,*SpecTcl - User's Guide* October 28, 2003. NSCL. http://docs.nscl.msu.edu/daq/spectcl/users_guide.htm

[2] D. Bazin,*Tree Parameters* 2005. NSCL. http://docs.nscl.msu.edu/daq/appnotes/TreeParameter.html

[3] N. Frank, *Sweeper Isotopic Separation* 2005. NSCL. `/projects/proj1/mona/reports`

[2] This button is used with both beam line tracking PPAC's or CRDC's.

[4] A. Ratkiewicz, W.A. Peters, *Time Sorting Pseudos* 2005. NSCL.
`/projects/proj1/mona/reports`

[5] J. Miller, M. Strongman, L. Elliott, D.B. Hecksel, M.M. Kleber, P.J. Voss, T. Pike, R. Pepin, A. Ratkiewicz, W.A. Peters, *MoNA Calibration* 2005. NSCL.
`/projects/proj1/mona/reports`

[6] A. Ratkiewicz, W.A. Peters, *MoNA SpecTcl Guide* 2006. NSCL.
`/projects/proj1/mona/reports`

[7] K. Yoneda, W.A. Peters, *MoNA Fitting Codes* 2004. NSCL.
`/projects/proj1/mona/reports`