



# Time Sorting Pseudo Codes for SpecTcl

A. Ratkiewicz & W. A. Peters  
National Superconducting Cyclotron Laboratory

March 8, 2006

## 1 Getting Started

This manual is intended to instruct the reader as to the proper use of the time sorting pseudo codes (pseudos). It is intended to supplement the *Mona SpecTcl Guide*[3] and the online *SpecTcl User's Guide*[1]. Please review these documents, as the concepts in them will not be readdressed.

### 1.1 Motivation

If you have created hit spectra in SpecTcl<sup>1</sup>, parameters (and spectra) exist for time of flights from the beamline scintillator to MoNA. These belong to the (TOF\_hit\_n) family of spectra and parameters, where  $n \in [1, 20]$ .

The (TOF\_hit\_n) family of parameters is filled arbitrarily; one should no more believe that the time of flight value in (TOF\_hit\_1) is more likely to be that of the first hit in MoNA than is the value in (TOF\_hit\_18). The object of this code is to sort the ToF values in order from first to last. A simple modification to this script will allow us to insist that all ToF values have an associated Q value (we can also specify the value that Q takes, if we like).

Once we have an ordered list of ToF values, we will record the indices of the parameters containing these values, and use this index list to reorder every other parameter family according to its position in time.

For the purposes of this manual, there are three time sorting scripts: a basic script (requires that the event has some real hits), a neutron window script (requires that the event occurs in the and a Q script (which has all the requirements of the basic script, but demands that there be some Q associated with each ToF).

### 1.2 The Basic Script

Following is the basic sorting script<sup>2</sup>. We'll discuss each line.

```
1 set paramnum 7000 parameter -new tof_val $paramnum ns; incr paramnum
2 pseudo tof_val $TOF_HIT_CPP
3 {
4   for {set i 1} {$i <= 20} {incr i}
```

---

<sup>1</sup>See *Mona SpecTcl Guide*[3]

<sup>2</sup>It is important to note at this point that SpecTcl does not like whitespace in its pseudos (and I haven't yet found a way to convince it that it should), so the above code is written on one line, which makes debugging interesting. Keep this in mind if you must edit the script.

```

5   {
6     if { [set TOF_hit_${i}\isValid] && [set TOF_hit_${i}] > -51}
7     {
8       set numval $i;
9       set x [set TOF_hit_${i}];
10      lappend big_t_list $x
11    }
12    else
13    {
14      set i 21
15    }
16  };
17  if {![info exists numval]}
18  {
19    set numval 0
20  }
21  else
22  {
23    global sorted_t_list;
24    set sorted_t_list [lsort -real -increasing $big_t_list];
25    global index_list;
26
27    if {[info exists index_list] == 1}
28    {
29      unset index_list
30    };
31    set p 0;
32    set k 0;
33    set index_length [llength $sorted_t_list];
34
35    while { $k < $index_length }
36    {
37      if {[lindex $sorted_t_list $k] == [lindex $big_t_list $p]}
38      {
39        lappend index_list[expr $p + 1];
40        incr k;
41        set p 0
42      }
43      else
44      {
45        incr p
46      }
47    }
48  };
49  global test_val;
50  set test_val $numval;
51  return $numval
52 }

```

### 1.3 Basic Script: Discussion

1: We create a variable, `paramnum`, and set it to 7000. We will use `paramnum` as the parameter id, incrementing it in future parameter definitions.

2: Create a new parameter, `tof_val`, with id `paramnum` and nanoseconds as units<sup>3</sup> (ns). Then increment the parameter id (`paramnum`).

3: Create a pseudo, `tof_val`, with the elements of the list `TOF_HIT_CPP` as arguments.

5: We want to sort every existing `TOF_hit_n` parameters for each event, keeping in mind that we have a maximum of 20 `TOF_hit_ns` for each event. Thus, we make a `for` loop, with `i` as the index, `i` running from `i=1` to `i=20`. We will use `i` as the index of `TOF_hit_i`.

7: Inside the `for` loop, we use an `if` statement check that the  $i^{\text{th}}$  iteration of the parameter has a value for this event (`[set TOF_hit_$$i\isValid]`) and that this value is physical (`[set TOF_hit_$$i > -51]`).

9-11: If the `if` statement is satisfied, we set our running total of real events (`numval`) equal to the index, (`i`). Next (10) we put the value of the current `TOF_hit_i` parameter into a variable, (`x`). Finally, we append the value of `x` (`TOF_hit_i`) to a list (`big_t_list`). We leave the `if` statement from line 7.

13-17: If line 7's `if` has failed, we set the index (`i`) to 21, effectively terminating line 5's `for`. Note that since this `for` loop is over, the closing bracket (line 17) has a semicolon after it.

18-21: We check to make sure that there are some valid events (`if {[info exists numval]}`). If none exist, then neither will `numval`, so we create it and set it to 0 so we can return it; if the event fails this `for` loop, we're done.

24: If 18's `if` statement fails, then there are some valid `TOF_hit_n` parameters, so we make a global list (`sorted_t_list`)<sup>4</sup>

25: At this point, `big_t_list` contains the values of all existing `TOF_hit` parameters for the event, so we set the global list we just created (`sorted_t_list`) to be an ordered from least to greatest version of the values in `big_t_list`.

26: Ultimately (as I've mentioned) we want to know how the `TOF_hit` family of parameters are ordered in time with respect to the indices, so we make a global list to contain these indices (`index_list`).

28-31: If `index_list` exists already, making it global (line 26) will put the last valid event's values into it. Thus, we clear it (`unset index_list`) if it exists.

The bulk of the rest of the code is a sorting algorithm:

32-34: we make two new indices, (`p`, `k`), and set them both to 0. We create a variable (`index_length`) containing the length of the sorted list of ToF values (it should be the case that `index_length = numval`).

36-48: This is a basic sorting algorithm whose structure isn't really determined by the eccentricities of `SpecTcl`. I don't feel compelled to discuss it, suffice to say that it gives us a time ordered list of indices

---

<sup>3</sup>I could just as easily use elephants as units; this declaration merely puts "ns" on the bottom of the associated spectra

<sup>4</sup>Global lists are accessible outside of the pseudo in which they were created. Since a pseudo is unwilling to return a list or an array, creating global lists is the best way to get more than one value from a pseudo. It is important to note that `global listname` does not create a list called `listname`, it merely says that if such a list is created, it will be a global list.

(sorted\_t\_list), which is what we've been after all along. Now that we have this, we can produce time ordered versions of every other parameter.

52: We return numval, so referring to the tof\_val parameter in any other pseudo will give us the number of valid hits in an event after sorting.

## 1.4 Neutron Window Script

The neutron window script is a slight modification of the basic script; we replace line 5 in the basic script with

```
5 if {[set TOF_hit_$i\isValid] && [expr int([set TOF_hit_$i])] > 50
6 && [expr int([set TOF_hit_$i])] < 100}
```

So the Neutron Window script accepts only values for TOF\_hit\_i that are between 50 and 100 ns. If your neutron window varies, change the upper and lower limits here accordingly.

## 1.5 Q Script

To make sure that a Q value exists for each hit, add

&&[set Q\_hit\_\\$i] > 0.99 to the argument of the if statement of line 5 in either of the sorting scripts.

# 2 Generating Sorted Parameters

Since we have a time ordered list of indices, we can produce ordered versions of any parameter in much the same way. I'll discuss a single example in detail, then I'll mention how to modify this example to produce a sorted version of any other parameter.

## 2.1 Producing Time Ordered Parameters

```
1 set X_HIT_TCL {X_hit_1 X_hit_2 X_hit_3 X_hit_4 X_hit_5 X_hit_6
2 X_hit_7 X_hit_8 X_hit_9 X_hit_10 X_hit_11 X_hit_12 X_hit_13 X_hit_14
3 X_hit_15 X_hit_16 X_hit_17 X_hit_18 X_hit_19 X_hit_20 tof_val}
4
5 parameter -new X_sorted_1 $paramnum cm; incr paramnum
6
7 pseudo X_sorted_1 $X_HIT_TCL
8 {
9   if {$tof_valisValid && $tof_val >= 1}
10  {
11    global index_list;
12    set val [lindex $index_list 0];
13    set x [set X_hit_$val];
14    return $x
15  }
16  else {return -303}
17 }
```

1-3: We make a list (`X_HIT_TCL`) with `X_hit_n`,  $n = 1..20$  as arguments. Such a list must be made for every parameter we are planning on ordering, because the first time ordered parameter could be any unsorted parameter.

5: We declare a new parameter (`X_sorted_1`) in units of centimeters (cm).

7: Create a pseudo (`X_sorted_1`) with the elements of the list `X_HIT_TCL` as arguments. Note that this list also includes `tof_val`.

9: `if` statement. We want `tof_val` to exist and (since this is `X_sorted_1`) be greater than 1. When we're looking for `X_sorted_n`, we want  $\text{tof\_val} \geq n$ .

11: We give this pseudo access to the global list of indices (`index_list`).

12: We set a temporary index (`val`) to the value of the  $0^{\text{th}}$  element.<sup>5</sup>

13-14: We find use the time ordered index to refer to the original parameter, at `X_hit_$val`, and return that value.

16: If the `if` statement in line 9 fails, we return -303, which is outside of the range of physical values for every standard parameter.<sup>6</sup>

To generalize this example to produce other parameters, create a list for all the values of the parameter (analogous to the `X_HIT_TCL` list), a new parameter, change the pseudo name and the name of the parameter in line 13 of the above example to reflect whatever you want to calculate.

## References

- [1] R. Fox, *SpecTcl - User's Guide* October 28, 2003. NSCL.  
[http://docs.nsl.msui.edu/daq/spectcl/users\\_guide.htm](http://docs.nsl.msui.edu/daq/spectcl/users_guide.htm)
- [2] 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](#)
- [3] A. Ratkiewicz, W.A. Peters, *MoNA SpecTcl Guide* 2006. NSCL.  
[/projects/proj1/mona/reports](#)
- [4] W.A. Peters, *Tandem SpecTcl Guide* 2006. NSCL.  
[/projects/proj1/mona/reports](#)

---

<sup>5</sup>In general, `val` is one less than the index

<sup>6</sup>If you are writing pseudos based on the sorting scripts, be careful to check that your arguments do not take this fail value; checking validity (`isValid`) on a calculated parameter (such as these) is of limited utility.