Eurogam Sort Reference Card

(for version 2.0)

Setting up

There are three steps involved in getting a sort program running.

A data file has to be created using any text editor. The data file has to contain sections as described in File Sections below.

Secondly, the complete description file must be passed through the Sort Compiler, see elsewhere.

File Sections

***formats** ... see Adc Definitions
<adc-definition-statements>
***data** ... see Data Definitions
<data-definition-statements>
***spectra** ... see Spectrum Definitions
<spectrum-definition-statements>
***commands** ... see Command Definitions
<command-statements>
***runfiles** ... (Offline only) tape files
<tape-file-definition-statements>
***specfiles** ... (Offline only) spectrum files
<spectrum-file-definition-statements>
***finish** ... File terminator

Parts of the description may be included from separate files ...

include <filename>

Adc Definitions

This is the first and possibly most important section of information required. It defines the maximum set of data acquisition parameters (ADCs, TDCs, etc.) to be expected in each event to be processed. Each parameter is labelled by an “address”.

<adc-name> <address>
<group-name> [<group-range>] (<item-list>)

Data Definitions

Definitions of variables used in the commands section. These variables are accessible interactively during the running of a sort.

word ⟦ <name> ⟦= <integer-value>⟧ ⟧r
long ⟦ <name> ⟦= <integer-value>⟧ ⟧r
float ⟦ <name> ⟦= <floating-point-value>⟧ ⟧r

gainword <name> <a> <b> <c>

gainarray <name> <n-gains>
⟦ <a> <b> <c> ⟧r=n-gains

gatemap mask <name> <ngates>
<16-bit mask>r=ngates

gatemap 1d <name> [<xrange>] <ngates>
(<low-limit> <high-limit>)r=ngates
gatearray <name> <ngates>
(<low-limit> <high-limit>)r=ngates

gatemap 2d <name> [<xrange>,<yrange>] <ngates>
(<xy-coordinate-pairs>)r=ngates

The **valuearray** command allows 1D,2D,or 3D arrays to be initialised with data provided in a free-format ordered datalist. The ordering is normal Fortran convention of x moves faster than y,moves faster than z.

valuearray <array-name> <xbase> <xrange>
⟦<ybase> <yrange> ⟦<zbase> <zrange>⟧⟧
<data-list>

Arrays can be written and read using Arithmetic Commands.

Spectrum Definitions

The ***spectra** section defines spectrum names and dimensions etc.

<spec-name> <nchns>	(1D,16bit)
<spec-name> <nchns> **32**	(1D,32bit)
<spec-name> <xchns> <ychns>	(2D,16bit)
<spec-name> <xchns> <ychns> **32**	(2D,32bit)
<spec-name> <nchns> **2D**	(2D,16bit)
<spec-name> <xchns> <ychns> <zchns>	(3D,16bit)
<spec-name> <xchns> <ychns> <zchns> **32**	(3D,32bit)
<spec-name> <nchns> **3D**	(3D,16bit)

where <spec-name> can be array of spectra ... NAME[1:10]
The **3d** variant is symmetrized (1/6 of cube).

Command Definitions

Commands all start on a new line except where indicated. For example, a simple command following an **if** statement.

<...-name> ... upto 8 characters (alphanumeric plus _)
<expr> ... see Arithmetic Commands

<parameter> is <adc-name> or <group>.<item-name>
<sortword> is one of ... word,long,float

Spectrum Access Commands

The 1D,2D,and 3D spectra defined in the ***spectra** section can be incremented and decremented using the following commands.

inc <spec-name> (<xchan>)
dec <spec-name> (<xchan>)

inc <spec-name> (<xchan> , <ychan>)
dec <spec-name> (<xchan> , <ychan>)

inc <spec-name> (<xchan> , <ychan> , <zchan>)
dec <spec-name> (<xchan> , <ychan> , <zchan>)

where <xchan>,<ychan> and <zchan> can be either an <expr> or <list-name>

Each of the above spectrum update variants may have an extra keyword field allowing indexing into an array of spectra. E.g. typical index could be **$gate** or **$groupx**.

inc <spec-name> (<xchan>) **indexed** <expr>

The following command increments channel if bit set in word.

incbits <spec-name>(<expr>) **offset** <integer-offset>

List Processing Commands

Lists are a useful way of applying commands to a selected set of similar <sortwords>. A list must first be created, and then used (see spectrum updates, flow control).

createlist <list-name> <parameter>r
createlist <list-name> <group.item>

Some condition testing commands can generate a revised list of those items not involved in the current test. Extract can be used to obtain a fixed number of sortwords from the list.

extract <list-name> <word>r ⟦**ordered**|**reversed**⟧

Loopextract loops over command(s) in brackets for every combination of “n” parameters in the list.

loopextract <list-name> <word>n ⟦**ordered**|**reversed**⟧
{
<one-or-more-commands>
}

Flow Control Commands

A series of commands that affect direction of flow. Various conditions may be applied to data parameters. In the following list there are, arithmetic comparisons,1D filter tests,1D veto tests, 1D mask tests and 1D/2D gate list tests.

```
if <test> <statements> ⟦ else <statements>⟧
where
<statements>      = <single-command>
                  or {
                       <one-or-more-commands>
                     }
<test> =   <sortword> valid
           <expr1> eq|ne|ge|le|gt|lt <expr2>
           <expr> passes (<lolimit> <hilimit>)
           <expr> fails (<lolimit> <hilimit>)
           <expr> maskedby <16bit-mask>
           <expr> gatedby <1d-gate-record>
           <xexpr> <yexpr> gatedby <2d-gate-record>
```

The following is the syntax for word-list tests. The same set of tests as above apply. The **loopfail** section is only executed if all tests in the list fail.

```
loopif <test> <statements> ⟦ loopfail <statements>⟧
where
<test> =   <list-name> valid
           <list-name> eq|ne|ge|le|gt|lt <expr>
           <list-name> passes (<lolimit> <hilimit>)
           <list-name> passes <gate-array>(<index>)
           <list-name> fails (<lolimit> <hilimit>)
           <list-name> fails <gate-array>(<index>)
           <list-name> maskedby <16bit-mask>
           <list-name> gatedby <1d/mask-gate-record>
           <xlist> <ylist> gatedby <2d-gate-record>
```

Gate tests set sortword **$gate** to gate number passed
All tests set sortword **$wordx** to current test word from list
2d gate tests similarly set **$wordy**
$groupx and **$groupy** return group number of current word
Resultant sub-lists are available in **$listx** and **$listy**
group(<sortword>) returns group part of address
item(<sortword>) returns item part of address

The following is the syntax for executing different sets of commands dependent on 1 or 2 values calculated.

select (<expr>)
⟦(<value>) <statements>⟧r
select (<expr1>,<expr2>)
⟦(<value1>,<value2>) <statements>⟧r

To remove whole group from current event ...

invalidate <group-name>

To break out of processing current event ...

endevent

To directly transfer command execution ...

goto <label-name>
label <label-name>

Looping Command

Equivalent of a do loop. The loop counter is available as sortword **loop**.

```
doloop from  <integer> to  <integer> step  <integer>
  {
  <one-or-more-commands>
  }
```

Arithmetic Commands

Expression defaults to integer unless resultant <sortword> pre-defined as floating point in ***data** section. Upto 6 levels of parenthesis are allowed. Fortran operator precedence.

<**sortword**> = <expr>
<**array-element**> = <expr>

Integer operators:	+ − ∗/ <> & **ior xor not neg abs**
Floating operators:	+ − ∗/ ∗ ∗ **neg abs**
Floating functions:	**sqrt exp log log10 sin cos tan** **asin acos atan**
Other functions:	**number**(<list>), **nbit**(<expr>), <array-element>,<word1>:<word2>

where
<array-element>=<array-name>(<x>)
<array-element>=<array-name>(<x>,<y>)
<array-element>=<array-name>(<x>,<y>,<z>)
and <x>,<y> and <z> are each <expr>

The sortword **random** can be used to obtain a floating point random number between 0.0 and 1.0

Gain Command

A special command that applies a gain adjustment to the specified parameter using the function $(a + bx + cx\hat{}2)$. A **gainword/array** must be defined in ***data** section.

gain <sortword> <gain-record-name>
gain <group-name> <gain-array-name>

Subroutines

Subroutine calls are allowed with upto 8 arguments. Any command may then be included up the return command (see below).

call <routine-name> (⟦<arg>⟧r)
routine <routine-name> (⟦<arg>⟧r)

Returns may be controlled by placing one of the following commands at the end of the routine.

end	returns to command after call
endevent	terminates processing event

Writing New Events

User specified events may be output using the following commands.

output <stream-number> **event**
output <stream-number> <sortword>r
output <stream-number> <list-name>r

Writing Sortfiles

An example:

```
*formats
trigger [255]   (type,mraw)
ge [1:32,34,71] (e1,e2,tac,hit)
*data
gatemap 1d gate1[0,4095] 3
(100 110) (200 210) (300 310)
include ~/gains/current-e1
*spectra
ge1 4096 2d
*commands
if trigger.type ne 1 endevent
gain ge.e1 e1gains
createlist list1 ge.e1
loopif list1 gatedby gate1 {
  extractall $listx  a b
    {
    inc ge1(a,b)
    inc ge1(b,a)
    }
  }
*finish
```

Produced by Janet Sampson and John Cresswell, April 1991 v2.0
Contact SAMPSON or CRESSWELL @ UK.AC.LIV.PH.GB

For copies of the Liverpool Sort Program User Manual ...
extract document EDOC033 from the NSF document server.