# USING THE QPOE I/O INTERFACE

This appendix fills in some details on what the QPOE interface can do; what it can operate on, processes that can be executed, and more complete description of the syntax for interface expressions. It is not exhaustive, and unfortunately, there doesn't seem to be any really complete description of the interface that is meant for end users. If you encounter problems or have further questions while processing your EUVE data, please send them to the EGO Center, at egoinfo.cea.berkeley.edu. The EUVE science and programming staff will support Guest Observer's efforts as completely as possible.

## A.1 QPOE File Headers

QPOE file headers can contain quite a lot of information, packaged in several different formats. As noted in the *EGO Center Data Products Guide* and in section 2.2 of this *User's Guide*, a partial representation of QPOE file headers can be obtained with the tasks **imhead** and **qpmedit**. To understand the operation of the the QPOE I/O interface, it is helpful to have a more complete picture of how non-event information is stored and maintained in QPOE files. This section describes QPOE file header parameters which, although they look the same in the output from **imhead**, are different from the header cards in FITS files or IRAF images This information is not necessary for doing most standard data reduction on EUVE QPOE files, but may prove useful to GO's who process their QPOE's with tasks in the **xray** package, or who wish to write their own QPOE-compatible IRAF scripts. Review the general information about QPOE files in the *EGO Center Data Products Guide*, Chapter 2, if some of the terms used are not familiar to you.

### A.1.1 Types of QPOE header parameters

QPOE header parameters can be separated into several basic categories:

- **Scalars** – The simplest parameters contain a single number or a single ASCII character. They are read directly by **imhead** and the literal values are printed in the output as if they were IRAF image cards.
- **Vectors** – Unlike IRAF images, QPOE files may have header parameters with two or more elements: vector parameters. The individual vector elements can contain numbers, ASCII characters, or more complex structures. Every QPOE file has a character vector called `EVENT`, containing attribute names and

definitions, which defines the structure of the event record. The event list itself is a very long, complex vector called `EVENTS`.

These two vectors appear in different forms in the output from **imhead**, and illustrate a basic limitation of **imhead** in dealing with vectors. **imhead** can print QPOE vector parameters that are character strings, and not too long, but when it encounters a non-character vector, it cannot print the literal value. Instead **imhead** prints the vector type, and the size in some appropriate units, using the usual IRAF data types: b=boolean, c=character, d=double, i=integer, l=long, r=real, x=complex[1]. So if the vector appears as

```
NAME = r[010]
```

it is an array of 10 real numbers. EUVE QPOE files do not include many non-character vector parameters as of this writing, so they are all printed entire except for some special cases.

- **Special Classes of Parameters** – In addition to scalars and the usual types of vectors, there are three other types of parameters which may be included in the QPOE file header. In EUVE QPOE files they are always vectors, though this is not a general requirement.

  1. **macros** – Macro definitions are stored as vectors of ASCII characters. Although the value is technically an ASCII string, **imhead** cannot print the value, because the vector is not of type `c`. Instead the task produces the entry `macro[#chars]`, with the number of characters, followed by the string `'macro definition'`. Use the task **qpmedit** (section 2.2.2) to examine the contents of macro definitions. The standard EUVE macros are more fully described in Chapter 4 of the *Data Products Guide*.

  2. **opaque vectors** – These are parameters that the QPOE interface cannot interpret for use. They may be intended for use by some other task in IRAF. **imhead** only gets the type and the number of bytes in the value of the parameter, with a short label that describes the contents. All EUVE QPOE's contain a `qpwcs`, which is a special case of opaque parameter. The `qpwcs` vector stores the spectral world coordinate system (WCS) that maps the attributes x,y to lambda,theta, in an internal binary format. The parameter is listed as `opaque` in the **imhead** output, but the QPOE interface also translates its elements into a set of header cards that specify the WCS in the usual IRAF format. The interface performs this translation so that when the interface passes the QPOE file to an image task, the task have a WCS to work with. The IRAF document *The IRAF/NOAO Spectral World Coordinate Systems*, listed in Appendix D, explains the IRAF WCS notation.

  3. **user-defined parameters** – These are in fact usually defined by the program that creates the QPOE file. Again, **imhead** cannot print the value; instead it prints `name[#elements]`, followed by a short descriptive label. The most

[1] "c" only appears in the case of an empty character vector, as c[000].

important example of a user-defined parameter is the event list itself. The vector `events` is made by the "user" **cep**, and has elements of a type defined by the value of `event`. The number of elements printed by **imhead** is the number of events in the event list.

### A.1.2 Reading the imhead display

When a QPOE file is processed by **imhead**, the output has a number of lines which appear identical to those produced by IRAF image header cards:

```
CAPSNAME = some string or numerical value
```

Other lines, like those produced by the QPOE macros, show only some labels and numbers; their literal values cannot be printed. **imhead** truncates all keywords to eight characters and displays them in capitals, although QPOE header parameters are not subject to these limitations like the image cards.

Unfortunately, **imhead** is the only tool in IRAF for examining most of the QPOE header parameters. Table A-1 shows each of the parameters in a standard EUVE QPOE file, in the order they appear in the **imhead** long output. We have identified each parameter with both the label printed by **imhead** and the internal QPOE name; the latter are often spelled the same, but are not capitalized. On the other hand, parameters which are inserted by **cep** for compatibility with IRAF image tasks *are* all-capitals. Any time a header parameter is used as input to a QPOE task, it must be spelled as it is stored in the file.

The table shows the type of each parameter as well as some information about how the **imhead** "value" is produced. All the entries except `BLOCKFAC` are from a part of the QPOE called the symbol table, having been put there by **cep** when it created the QPOE. Many of the lines that translate most literally, like the EUVE global observation parameters[2], are put in the symbol table for the user's information only.

The values of some **imhead** entries are generated "on the fly" by the QPOE interface when it opens a QPOE file for processing as an image. This is because the symbol table is also used to pass information about the QPOE to IRAF image tasks, with the QPOE interface operating on the table entries to package them into "card-like" values. The values may be printed (as by **imhead**) or used in some other way by another task. Some examples:

- The parameters `naxes` and `axlen#` are required by many IRAF **image** tasks.
- The opaque vector `qpwcs` is stored in binary and expanded into a series of WCS cards by the QPOE I/O interface.
- `BLOCKFAC` is the value of a QPOE I/O parameter that sets a default bin size.

[2]Described in Appendix B of the *EGO Center Data Products Guide*.

Table A-1: EUVE QPOE file header parameters, as generated by **imhead**.

| imhead label | QPOE name | param type | origin | comments |
|---|---|---|---|---|
| `Created` | cretime | scalar | inserted by **cep** | not FITS card format |
| `Last modified` | modtime | scalar | inserted by IRAF | not FITS card format |
| `min` | datamin | scalar | generated by QPOE I/O | minimum data value |
| `max` | datamax | scalar | generated by QPOE I/O | maximum data value |
| `NAXES` | naxes | scalar | generated by QPOE I/O | 2 axes for EUVE QPOE's |
| `AXLEN1` | AXLEN1 | scalar | generated by QPOE I/O | 2048 for EUVE QPOE's |
| `AXLEN2` | AXLEN2 | scalar | generated by QPOE I/O | 2048 for EUVE QPOE's |
| `BLOCKFAC` | blockfactor | scalar | generated by QPOE I/O | QPOE I/O parameter |
| `QPFILT01` | PFILT01 | char vector | generated by QPOE I/O | filter used whenever the QPOE is opened. Up to 32 cards. |
| `WCSDIM` | N/A | scalar | generated by QPOE I/O | element of `QPWCS` |
| `CTYPE1` | N/A | char vector | ” | element of `QPWCS` |
| `CTYPE2` | N/A | char vector | ” | ” |
| `CRVAL1` | N/A | scalar | ” | ” |
| `CRPIX1` | N/A | scalar | ” | ” |
| `CRPIX2` | N/A | scalar | ” | ” |
| `CDELT1` | N/A | scalar | ” | ” |
| `CDELT2` | N/A | scalar | ” | ” |
| `CD1_1` | N/A | scalar | ” | ” |
| `CD2_2` | N/A | scalar | ” | ” |
| `LTM1_1` | N/A | scalar | ” | ” |
| `LTM2_2` | N/A | scalar | ” | ” |
| `WAT0_001` | N/A | char vector | ” | ” |
| `WAT1_001` | N/A | char vector | ” | ” |
| `WAT2_001` | N/A | char vector | ” | ” |
| `QPWCS` | qpwcs | opaque | inserted by **cep** | world coordinate system |
| `ORIGIN` | ORIGIN | char vector | inserted by **cep** | global observation info |
| `OBSERVAT` | OBSERVAT | char vector | ” | ” |
| `TIMESYS` | TIMESYS | char vector | ” | ” |
| `EQUINOX` | EQUINOX | scalar | ” | ” |
| `JDREF` | JDREF | scalar | ” | ” |
| `TIMEUNIT` | TIMEUNIT | char vector | ” | ” |
| `RA_UNIT` | RA_UNIT | char vector | ” | ” |
| `DEC_UNIT` | DEC_UNIT | char vector | ” | ” |
| `HIST001` | HIST001 | char vector | ” | ” |
| `REFVERS` | REFVERS | char vector | ” | ” |
| `EGOCSVER` | EGOCSVER | char vector | ” | ” |
| `DATE-OBS` | DATE-OBS | char vector | ” | ” |
| `TIME-OBS` | TIME-OBS | char vector | ” | ” |
| `DATE-END` | DATE-END | char vector | ” | ” |
| `TIME-END` | TIME-END | char vector | ” | ” |
| `INSTRUME` | INSTRUME | char vector | ” | ” |

Table A-1, Cont'd: EUVE QPOE header parameters

| imhead label | QPOE name | param type | origin | comments |
|---|---|---|---|---|
| `TELESCOP` | TELESCOP | char vector | inserted by **cep** | global observation info |
| `PI_NAME` | PI_NAME | char vector | " | " |
| `OBJECT` | OBJECT | char vector | " | " |
| `RA_OBJ` | RA_OBJ | char vector | " | " |
| `DEC_OBJ` | DEC_OBJ | char vector | " | " |
| `PROPID` | PROPID | char vector | " | " |
| `DEFATTR1` | defattr1 | char vector | " | defined as: integral time:d |
| `TIME` | time | macro def'n | " | EUVE attribute name |
| `X` | x | macro def'n | " | " |
| `Y` | y | macro def'n | " | " |
| `LAMBDA` | lambda | macro def'n | " | " |
| `THETA` | theta | macro def'n | " | " |
| `DX` | dx | macro def'n | " | " |
| `DY` | dy | macro def'n | " | " |
| `XS-EVENT` | XS-EVENT | char vector | " | for use with **xray** |
| `XS-SORT` | XS-SORT | char vector | " | for use with **xray** |
| `EVENT` | event | user defined | " | char vector defining event format |
| `EVENTS` | events | user defined | " | the event list |
| `DISPAXIS` | dispaxis | scalar | " | 1 for dispersion along columns |
| `DC-FLAG` | dc-flag | scalar | " | 0 for linear WCS |
| `EVENTSOR` | EVENTSOR | scalar | " | number of events processed |
| `EVENTSIR` | EVENTSIR | scalar | " | number of events rejected |
| `(DET)_GT` | (det)_gt | macro def'n | " | time filter |
| `EARTH_GT` | earth_gt | macro def'n | " | time filter |
| `(DET)_ADCT` | (det)_adct | macro def'n | " | time filter |

- The parameter `defattr` is a special case; it's string value defines `exptime` as the integral of time over any time filter present when the QPOE is opened. When the QPOE event list is binned to make an image, the interface calculates the integral and places the result in a card called `exptime` in the image header. The `exptime` card is not displayed in the output from **imhead** because **imhead** does not require the event list to be binned.

### A.1.3 Editing QPOE header parameters

Only a few QPOE header parameters are accessible to users; the rest are created and used only by tasks that operate on QPOE's. Users can edit the default parameters, `deffilt` and `defmask`.

The default filter `deffilt` has the form:

```
deffilt = att1r=(string1),attr2=(string2)...
```

which is applied to the event list every time the QPOE file is opened. Default header parameters can be added, deleted, or edited using the task **ximages.qphedit**. Using **qphedit** is very similar to using **qpmedit**, which is described in section 2.2.2. One reason to do this is to use the **xray.xtiming** tasks, as discussed in section A.1.4 and example A.1. The task **qphedit** will allow any string to be placed in the QPOE file, but a valid `deffilt` must contain complete filter syntax. The default filter should contain attribute filters only; the **xray** tasks cannot interpret default filters with any of the function keywords, such as `block` or `rect`.

Tasks in the **xray** packages will ignore command line filters and use the `deffilt` instead. **euv** package tasks merge the default filter with any other filters present, which may produce invalid results due to a QPOE interface bug (see box on page 2-5). Special syntax can be used to negate a `deffilt` (see example A.1).

One can also use default region masks of the form:

```
DEFMASK = PLIO region specification
```

PLIO region masks are described briefly in the subsection on `mask` in section 2.1.3.

### A.1.4 Processing EUVE QPOE files with the XRAY package

EUVE QPOE files now contain the header parameters `XS-EVENTS` and `XS-SORT`, which provide compatibility with tasks in the **xray** packages. These include **xray.xtiming, xray.xspatial,** and **xray.xplot**. Type "help xray" for a listing of packages and topics. Another requirement for compatibility with most **xray** tasks is the addition of the `deffilt` header parameter, whose value is a default attribute filter.

If **xray** tasks are used on a file with no `deffilt`, or no time filter in `deffilt`, the task will run without error, but no time filtering will take place. This is true *even if a time filter is included on the command line.* The task will display the message:

```
No GTI records in file - no TIME update done
```

The note `(; no exp.)` will appear in the header of the resulting QPOE or image file, and the value of `exptime` will not be calculated correctly by the **xray** task. This is because **xray** tasks are written to look for filtering information in the QPOE file header parameters `deffilt` and `defmask`. It is not a characteristic of the QPOE interface itself. Tasks in the **euv** package can parse command line filters and a `deffilt`, or a combination, but then filter merging may become an issue (see the box in section 2.1.2).

Once a `deffilt` card has been inserted, this filter is always applied to the QPOE events when the QPOE is opened by the interface program, unless a special QPOE I/O syntax is used. This means that those who use tasks in the **xray** packages should exercise extra care when processing these QPOE's, as the `deffilt` will interact with any on-line filters on the same attributes when processing or making images. Because of the QPOE interface bug outlined early in section 2.1.2, *if a filter specified on the command line is applied to the same attribute as* `deffilt`, *the results may not be correct.* If, for example, both filters are applied to the `time` attribute, the resulting exposure time may be incorrect.

If `deffilt` can be used to hold all the exposure and data quality filters required, then processing the QPOE can be done without any other filters on the command line. But if filters other than `deffilt` are needed for some phases of processing, it is easiest to remove the `deffilt` with **qphedit** beforehand. See section A.2.2 to read about the QPOE I/O interface syntax for nulling the effect of `deffilt` on the command line.

One **xray** task which may be useful for GO's doing timing analysis is **xtiming.timsort**. This task resorts the QPOE file in time order, which makes subsequent time-filtering operations run much more efficiently. Although **timsort** is set up to operate most easily on PROS QPOE files, a complete resorted copy of an EUVE QPOE can be made by specifying the value of the task parameter `esize` as follows:

```
esize = "d:time,s:x,s:y,r:lambda,r:theta,s:dx,s:dy"
```

Note that each attribute to be included must be specified by both type and name, the latter of which is used as the column name in the new QPOE file.

## A.2 Additional Notes on QPOE Expression Syntax

The description of the QPOE interface given in section 2.1 should enable you produce valid basic QPOE expressions. While it is not complete, this section adds a bit more detail, and some variations of the QPOE expression syntax that some GO's may find useful.

### A.2.1 Multiple intervals and filters

In QPOE expressions with attribute filters, the filter string can be quite long and complex, as we have seen. The exact selection rule produced depends on punctuation as well as on the interval specifications.

#### Commas and parentheses

Generally, the commas between intervals function as logical OR's; any event in any of the comma-separated ranges passes this expression. So the filter expression:

```
dx=(5:10,11,16:17,20:)
```

Will select all events with a value of `dx` between 5 and 10, OR equal to 11, 16, OR 17, OR greater than 20. In an expression like:

```
sw.qp[dx=(5:10,11,16:17,20:),dy=(1:5),time=(sw_gt)]
```

the commas between the three attribute filters are ANDs; an event passes this set of filters only if it matches all three filter expressions.

Commas and parentheses can also be used to produce logical ANDs within a filter string for a more stringent criterion. For example:

```
'mw.qp[y=((900:1100),!(1018:1032)),x=((140:1780),!(1200:1400))]
```

will produce a background region equivalent to a 200-row strip in Y, excluding 15 rows where the spectrum falls, and a section in X with 200 columns, where a bright airglow line raises the normal background. The outer ( )'s delimit the filter string associated with the attribute, and the inner ones delimit the ranges (intervals) that are being ANDed together for each one. The resulting QPOE can be trimmed of its 0 "pixels" by using a `rect` mask as well.

In addition, when filters on different attributes are ANDed together, the interface applies the individual filters in the order in which they appear in the QPOE expression. This means users can save processing time by putting filters that eliminate the most events first. Then the interface doesn't waste as much time putting events through one filter only to have them eliminated by later ones. For example, in processing EUVE QPOE's, good times filters should generally come last, since they do not normally eliminate many events. On the other hand, the order of the attribute ranges that are ORed together within each filter is unimportant.

#### Examples of specialized QPOE expressions

1. Make an image of a QPOE in raw detector coordinates and block the resulting image by a factor of 2. The `key` expression comes first, and the blocking is then performed on the detector coordinates. The output image will be $1024 \times 1024$ pixels in size.

   ```
   im> qpmkim 'test.qp[key=(dx,dy),block=2]' test.imh applyc+ adjexp+
   ```

   All other parameters of **qpmkim** must be set in advance.

2. Filter the QPOE according to the x coordinate. The interface bins all events with an x value between 100 and 2000. The only events that pass the filter are those from the intervals (100–1200) and (1400–2000). Others are 0.

```
im> imcopy 'test.qp[rect=[100:2000,*],x=!(1200:1400)]' test.imh
```

3. Select a section of a Deep Survey QPOE file by filtering on RA and Dec:

```
im> imcopy 'sw1.qp[ra=(19.853:19.953),dec=(-210.10:-210.20)]'
sw_blkspec.imh
```

### A.2.2 Special punctuation for attribute filters

The QPOE I/O interface provides some variations on the '=' operator for combining or selecting among multiple QPOE filters, which may appear in `deffilt`, macro definition files (described below), and command line expressions. When the QPOE interface finds multiple filters associated with a single QPOE file, the default action is to merge (AND) all the filters, including filters on the same attribute, to form the logical intersection. As noted in section 2.1.2, the interface cannot be relied on to merge two or more filters on the same attribute reliably at this time (version **2.10.4** and earlier). The conventions we describe here are subject to the caveat that at present, combining two or more filters on any one attribute should still be done with the IRAF/EUV task **tfilter**. The following syntax can be used successfully if they do not imply merging filters this way.

One variant on `attribute =(string)` is `attribute:=(string)`. In the case of :=, any filters already entered for the named attribute, in `deffilt`, in a macro, or earlier on the command line, will be REPLACED by the new expression.

Another special case is the syntax `!attribute=(string)`. In this case, Any previously entered filter on *any* attribute will be cancelled and replaced by the part of the expression after the symbol `!`. This should not be confused with the use of `!` to negate an interval within a filter string.

---

**Example A.1 : Working around a `deffilt`**

In the particular case of processing QPOE files with the **xtiming** package, users may need to know how to include or exclude the default filter in `deffilt` with other filters. Suppose, for example, that a default filter has been added to a QPOE file, in order to resort two orbits' worth of events in time order with **timsort**. The parameter `deffilt` is added to the QPOE header with the task **ximages.qphedit**:

```
xi> qphedit sw deffilt time=(780800001.4596:780802014.6436,780805
525.9396:780807708.0837) add+ update+
```

When **imhead** is run on the file *sw.qp*, the following line appears:

```
DEFFILT = 'time=(780800001.4596:780802014.6436,780805525.9396:78080

7708.0837)'
```

This might not be quite the time filter needed for some other operation, such as making a new spectral image for the entire observation. To replace this filter with one using the good times macro, `sw_mygt`, in the QPOE, one could type:

```
eu> qpmkim 'sw.qp[!time=(sw_mygt),rect=[*,720:1299]]' swgt adjexp+
dead+ prim+
```

This would result in a standard sized ($2048 \times 580$) 2-dimensional image, using the GO's specification of good time intervals, instead of the `deffilt`, to filter the event list and calculate the value of `exptime`.

If, on the other hand, the GO has created a `deffilt` with filters on more than one attribute, and wishes to replace only part of the expression, a different syntax is used. The user may have placed a wavelength filter in the `deffilt` as well:

```
DEFFILT= 'time=(780800001.4596:780802014.6436,780805525.9396:780807
708.0837),lambda=(90:120)'
```

In this case the GO replaces the time filter with her own good times, while keeping the spatial filter for trimming the image, using the following QPOE expression in the command to **qpmkim**:

```
eu> qpmkim 'sw.qp[time:=(sw_mygt)]' swgt adjexp+ dead+ prim+
```

This results in only the time filter of `deffilt` being replaced; the wavelength filter is still applied to produce an image of part of the spectrum.

Users should bear in mind that any spatial filters other than `rect`, including PLIO region masks, will not make the image smaller, but produce a $2048 \times 2048$ image with zero values outside the filter or mask region.

## A.3 Macro Types and Hierarchies

Macros can be defined several ways and in several places, allowing for a hierarchy of precedence. We will discuss the major types of macros and how to use them, but this section is not meant to be an exhaustive treatment of the possibilities. There are three major types of macro references, which are summarized in table A-2, and defined in more detail below.

Table A-2: Form of QPOE file macro references

| macro type | action | examples |
|---|---|---|
| text substitution | `macro` expanded to part of expression from definition | 'sw.qp[`time`=(1:300,900:1200)]'<br>`time`⇒ d0 |
| file inclusion | *file* is opened to find `macro` definition | 'mw.qp[@*timefilter*]'<br><br>*timefilter*⇒ time=(1:30,90:120,180:210) |
| command subst. | output of the named **task** inserted to complete expression | 'lw.qp[time=('**tfilter**')]'<br><br>**tfilter**⇒ 1:300,900:1200,1800:2100 |

1. **text replacement:** This is by far the most common type of macro. When a defined macro name is encountered in a QPOE expression, the interface searches the QPOE file and any macro definition files (see section A.3.1) for a macro definition with the same name, and the name is replaced by the definition of the macro. For example, EUVE QPOE files have macros that assign a mnemonic name to each attribute. The macro "x" is defined in EUVE QPOE files as "s8", the short integer attribute beginning at byte 8 of each record. When the expression `[x=50:100]` is interpreted, the interface converts it to `[s8=50:100]` before it is applied to the event list.

2. **file include macro:** This looks the same as the IRAF mechanism for suppling a list of arguments from a file, *arguments*, to a task parameter with "@arguments". Instead, the file should contain one or more complete keyword expressions. As always, the filename must include the complete path to the file from the current working directory. For example, a user who wanted to create a number of images from different time-slices of his QPOE file could use the command:

   ```
   im> imcopy 'mw.qp[@home$timeseg]' mw_orb30-33
   ```

   where the file *timeseg* is in the IRAF home directory, and contains a keyword expression of the form:

   time=(774683832.7481:774685492.6521,774689532.3321:774691192.2361,
   774695231.9169:774696886.7002...
   ⋮

   The QPOE interface converts all EOL characters to spaces. This example also makes use of text substitution macros: the string 'time' is a macro for '`d0`'. Up to 20 levels of nested macros are permitted.

3. **command substitution:** The macro reference is given as a `cl` command surrounded by quotes, which is replaced by the output of the command. For example the expression:

   ```
   [block=4,time='tfilter']
   ```

   where **tfilter** is a task which renders a table of start and end times into a set of time intervals in QPOE filter format (see section 2.2.3). The parameter file of the task to be run may be set up beforehand, or the task may be referenced with arguments, as in:

   ```
   [block=4,time='tfilter(sw_gt,my_gt mode=a)']
   ```

   This includes setting the task mode to `a` (for automatic), to suppress queries. Since there are not many commands that produce output in valid QPOE expression format, use of this type of macro is limited.

In all three macro types, the replacement text may contain embedded spaces, tabs, and newlines, each of which is converted to just one space on expansion. Recursion is also allowed, meaning that the replacement text may contain references to other macros, which will be expanded as encountered.

### A.3.1 Global macros and the *QPDEFS* file

The QPOE interface recognizes two categories of macros: "global" macros, which reside in a separate file and may be referenced in any QPOE expression, and "local" macros, which are stored in the QPOE file and are specific to that file. Macros may be defined in more than one place; for example, a macro called `time` may be defined in both a QPOE file and in an external file such as the standard file *QPDEFS*. If a macro of the same name is defined in more than one place, then a locally defined macro takes precedence over a global macro of the same name.

Global macros can be defined in external ASCII files, called macro definition files (MDF's). Such macros are global because they are available to any QPOE file expression. The interface looks at the `cl` environment parameter `qmfiles` to find files which contain global macros. The value should be a comma-separated list of files to be searched for macro definitions. IRAF virtual paths are allowed. There is no default setting for this variable; it must be explicitly set by the user. If multiple definitions of the same macro exist among the macro definition files available, the last occurrence, as determined by the order of filenames in `qmfiles`, is used. An existing local QPOE file macro with the same name takes precedence over a global macro in an MDF.

Definitions of macros in an MDF have a different form than macros installed directly into a QPOE file. Macros in MDF's must be defined with the syntax:

```
define macroname <replacement text>
```

with one such definition per line.[3] Remember that any MDF with macro definitions must appear in the list displayed by

```
eu> show qmfiles
```

for the macros to be applied. Macro definition files may also contain lines which set the values of QPOE interface parameters, as discussed in section A.4 below.

An example of a macro definition file exists in all IRAF environments, in the file */qpoe/QPDEFS*. It contains a set of QPOE interface parameter settings and some examples of macro definitions, but its contents are not required to run any of the IRAF/EUV software packages. It's just a handy template for making one's own macro definition files. You can copy the file into your IRAF startup directory and add definitions as you create global macros. If you use only your personal version of *QPDEFS* to store global macros, `qmfiles` can be set with the command:

```
cl> set qmfiles = "home$QPDEFS"
```

which can be added to your *login.cl* file.

When the QPOE interface encounters a macro name in parsing an expression, the QPOE file being accessed is searched for the macro definition. If it is not found, then the interface searches the macro definition files named in `qmfiles`. This is why local macro definitions take precedence. Be aware, however, that if you change the contents of a macro definition file during an IRAF session, the changes may not take effect until you reset `qmfiles` or restart IRAF.

**Note on MDF's for IRAF/EUV users**

The **euv** package contains a package-wide MDF, in *euv$lib/qp.defs*. Whenever someone starts the **euv** package, the variable `qmfiles` is checked – if it already has a value, it is not changed, but otherwise the value is set to point to the *qp.defs* file. Part of the function of this MDF is to expand the system ability to deal with long QPOE macros; an ability many GO's will need. Therefore, users who create new MD files should always include the standard **euv** file *euv$lib/qp.defs* in the list in `qmfiles` (see example A.2).

## A.4 QPOE Interface Parameters

The QPOE interface has a number of parameters that set assumed sizes for buffers, QPOE file buckets, and other I/O tools. These values are normally stored in a part of the QPOE file that is transparent to users. When an EUVE QPOE file

[3] This syntax is different from that used in file include macros–files included in [ ]'s must contain complete keyword expressions.

is converted to a FITS file using **qpwst** and **stwfits**, the list of interface parameter values is saved in a binary table extension. Compare the following list with the list of entries in *qp_AAA_qpparam.tab* in Appendix B, section B.5.1 of the *EGO Center Data Products Guide*, version **1.5.A**.

```
bucketlen     - bucket length, number of event records
cachesize     - of file descriptors in lfile cache
indexlen      - of hash entries in symbol table index
maxlfiles     - number of lfiles in datafile
pagesize      - size, bytes, of datafile
sbufsize      - symbol table size
stablen       - symbol table string buffer size, chars
progbuflen    - program buffer size (compiled instructions)
databuflen    - data buffer size (initialized data space)
nodeffilt     - the use of any default event filters
nodefmask     - the use of any default region masks
maxpushback   - characters pushback (for macro expansion)
maxfrlutlen   - full resolution lookup table length
maxrrlutlen   - reduced resolution lookup table length
lutminranges  - ranges required before a lookup table is used
lutscale      - factor to convert nranges to n LUT bins
blockfactor   - blocking factor for output pixel arrays
optbufsize    - I/O buffer size for IMIO access to QPOE file, chars
debuglevel    - debug level, 0 for no runtime debug messages
```

Each file carries a list of interface parameter values appropriate for processing that file, which is not seen in the output of **imhead**. Default values of QPOE interface parameters can be set globally by inserting the syntax:

```
set parameter value
```

in a user-defined macro definitions file, such as *QPDEFS*. Any such file must be included in the value of the environmental parameter `qmfiles`. Most of the time, most of these parameters are set at the system level, and need not be changed. However, the application in section A.4.1 (below) is one instance in which GO's may need to alter the values of some interface parameters.

### A.4.1 Accommodating long macros

One situation in which it may be necessary to reconfigure the QPOE interface is when one wishes to use a very long macro, such as one with a large number of short intervals. In terms of the usual default settings of interface parameters, a long macro means something over 8000 characters, or 64,000 characters with the **euv** package tasks installed.

A symptom of exceeding the limit is an error when trying to make an image from the QPOE file with the commands **qpmkim** or **imcopy**. The error message:

```
ERROR: Cannot open pixel file
```

is most common. It is also possible that there will be a different error message or no message at all. The command will finish unusually quickly when this problem arises, and the image produced will be incorrect or unusable. This problem will also occur with other commands, such as **display**. When the problem arises, the user can generally accommodate the large filter by adjusting some of the QPOE interface parameters. The QPOE interface is not designed for simple adjustment parameters by users, and GO's who do not have this problem will definitely want to skip the rest of this section.

It is difficult to devise values for the interface parameters which will work for all possible filters, so the values suggested below should be considered as starting points for your empirical research. To help you in this endeavor, we first, give an (optional) detailed explanation of the interface parameters involved, and follow this with a step-by-step recipe for changing the parameters. Please refer to Chapter 2 for definitions of terms regarding QPOE files with which you are unfamiliar.

#### Optional: Why long macros don't work

There are two stages at which a QPOE macro can be too large. The first is during the macro text replacement process. The maximum number of characters which can be pushed back into the input stream when replacing the name of a macro with its text is set by the parameter `maxpushback`. The default value for `maxpushback` is normally 8192 characters. When the **euv** package is loaded, this parameter is reset to 65536 characters. You can check the size of the macros in a QPOE file with the task **imhead**, as in:

```
im> imhead foo.qp long+
```

Find the macro name at the left, near the bottom of the listing. The number in square brackets is the size in characters. If this exceeds the value of `maxpushback`, you will get an error.

The obvious solution is to look at your macro lengths and set `maxpushback` to a value larger than the largest macro you will be using. Then this stage of the filtering process cannot fail. This may solve the problem, or it may result in a different error message from the next stage of the interface program.

The second time the filter can fail is when the expression is compiled. When reading a QPOE file, keyword and filter expressions are translated into an interpreted program and stored in an internal buffer. This buffer must be large enough to hold the program; the more intervals in the filters, the more space it needs. The buffer size is controlled by the parameters `progbuflen` and `databuflen`, with default values of 8192 bytes each. A related parameter is `maxrrlutlen`, which controls nesting of lookup-tables (don't worry about what this means) and has a default value of 1024.

The method for setting all these parameters to values other than their default is to edit one of the macro definition files, or MDF's (see section A.3.1), which are pointed to by the QPOE environment variable `qmfiles`. The default MDF supplied with the PROS **xray** package is called *QPDEFS*. Macro definition files may contain lines of the form:

```
set foo bar
```

where `foo` is the name of a QPOE interface parameter and "bar" is the value to which it is to be set. For example, if you look at the file *euv$euv.cl*, which is executed when you start the **euv** package, you will see the line:

```
set qmfiles = ("euv$lib/qp.defs")
```

which places the file *euv$lib/qp.defs* on the list of MDF's. If you look at *euv$lib/qp.defs*, you will see that it contains only two lines (the first is a comment):

```
\# set up the 64k buffer for qpoes macros
set maxpushback 65536
```

This is how the parameter `maxpushback` is increased from 8192 (its default) to 65536 characters when the **euv** package is loaded.

Unfortunately, because there are no tasks that will directly probe the length of macro programs, their data buffers, or their nesting habits, there is no easy way to predict the buffer lengths required for `progbuflen`, `databuflen`, and `maxrrlutlen` when using any particular filter. You will need to use trial and error to arrive at good values. What you must do to choose your own values is reset `qmfiles` to point to an MDF of your own which contains lines to change the values of all the parameters of interest to you. The step-by-step recipe given next explains how to do this.

---

**Example A.2 : Adapting the QPOE interface for large macros**

Here are step-by-step instructions for changing the parameters from inside IRAF:

1. Copy the file *euv$lib/qp.defs* to your IRAF home directory:

   ```
   cl> copy euv$lib/qp.defs home$qp.defs
   ```

2. Edit the file *qp.defs* in your IRAF `home` directory. Change the value for `maxpushback` to a number larger than your longest macro (if you don't know what this means, it is described further above). Add the three lines:

```
set    progbuflen       50000
set    databuflen       150000
set    maxrrlutlen      8192
```

to the file. The suggested values ought to work with quite large filters.

3. Change `qmfiles` to point to your file

   ```
   cl>  set qmfiles = "home$qp.defs"
   ```

   Be sure not to reload the **euv** package after this or you will overwrite the value of `qmfiles` with the **euv** package default value.

4. That should do it. Go to your data directory and try your large filter. If it still doesn't work, re-edit the file and increase `progbuflen` and `databuflen` to larger values. It shouldn't be necessary to increase `maxrrlutlen` further. If you need more information to choose appropriate values for the parameters, read the section above on the gory details.

If you need to use large filters all the time, you can place the command in step 3 into your *login.cl* file, where it will be executed every time you start IRAF. If you choose that option, and you also load the **euv** package in your *login.cl* file, be sure to set `qmfiles` after the **euv** package is loaded, to prevent overwriting `qmfiles`.

## A.5 Trouble-shooting your QPOE interface expressions

The QPOE interface can seem quite finicky and obstinate about accepting your expression syntax. Here is a short list of error messages, and some (incomplete) suggestions about may be wrong.

Unfortunately, it is also known that sometimes the QPOE interface produces some output, but not the right or expected output, and gives no error message whatsoever. You have probably already read about some of these possibilities elsewhere in this *User's Guide*. The best insurance against unknowingly using bad results from QPOE files is to test every product as carefully as possible and as soon as it is created. Address your unsolved problems with EUVE QPOE files to **egoinfo@cea.berkeley.edu**; we want to know about them, and you can usually get help.

PS: Don't forget to flush the IRAF process cache:

```
eu>  flprc
```

whenever an attempt to run any task fails.

? `ERROR on line 70:  QPEX: bad range specifier in event attribute filter`

There may be misspelled attributes or macro names. The [ ]'s around a rect specifier may be missing or replaced by ( )'s. This message can also appear when an **xray** task gets some input it isn't expecting. If using **xtiming.timsort**, it may not like the way you've specified the output event record in `esize`, or the number in `ssize` may be too small. It will also occur if there are non-attribute keywords in `deffilt`.

? `QPIO: unexpected token ':'`

Some ( )'s may be missing around some filter string with commas in it.

? `ERROR: Newline while processing string`

The single quote is probably missing from the end of a QPOE expression used on the command line.

? `ERROR: Cannot open pixel file`

This often means that the task was not expecting a QPOE, but an image file, because the ".qp" suffix was omitted from the input file name. However it may mean that a macro used in a QPOE expression is too long to be parsed by the interface– see section A.4.1.

? `ERROR: segmentation violation`

This may mean that your directories are nested too deeply, and you have created a path length to the QPOE (or other file) that exceeds the buffer size in IRAF.

# PLOTTING ST TABLE COLUMNS WITH TBPLOT.IGI

The interactive graphics interpreter in the **tables** package, **tbplot.igi**, is a versatile plotting tool capable of handling data in many IRAF formats. GO's are particularly encouraged to use **igi** in situations requiring a flexible graphic representation of data in ST table format. **igi** is capable of reading data from images or ST tables and allows the user to plot any two image coordinates or columns against each other. In this Appendix, we describe using **igi** to plot various columns of EUVE ST tables. This is a good alternative to using **dqselect** as a display tool, since **igi** reads the entire column instead of sampling the input. This means that, by adjusting plot limits for the data set and the display device, one can view at least a subset of the table data at full resolution. By contrast, display in **dqselect** is always limited to the sample of data points read in when the table is first loaded on startup.

One caveat: **igi** cannont handle double precision numbers. This means that to plot any table column with doubles, such as time columns in EUVE ST tables, the user must first add a column to the table with the values offset by the first value, and converted to real numbers. This is quite simple to do with tasks in **tables.ttools**:

1. Look at the table using **tread**, and record the first time stamp, $T_0$.
2. Use **ttools.tcalc** to add a new column to the table with the difference:

    timecolumn - $T_0$

3. Check the new column using **tread**. If, for example you want to plot an aspect table, with times every major frame, the new column should have the values 0, 1.024, 2.048, etc., with orbit daytime gaps occurring about every 3600 lines.

When you have done any necessary preparation on the table, load the **tbplot** package and start **igi**. This will put you in interactive mode with a status line at the bottom of the STDGRAPH screen.

```
tt> tbplot

  axispar dvpar igi pltpar sgraph

tp> igi
```

(switch to STDGRAPH)

```
STScI igi Version 24Jan94 -- type HELP for info, END to stop igi>
```

The task uses the standard commands available in the *mongo* graphics command interpreter, with extensions to take advantage of IRAF facilities like "snap". All the commands can be listed by typing the `help` command with no argument. Type `help command` to see the message for `command`. Help messages are directed to the text window; use two 'q' commands to return to the interactive mode.

The following lines give a basic script for loading the table *mkaspectds.tab*, which was produced for the Deep Survey pointing by the example parameters in figure 1-8. The table has had a real-number time column added, and the user first plots the roll angle column against this time. The left column shows command syntax, with comments on the right.

| | |
|---|---|
| `data mkaspectds` | Causes **igi** to locate the table. |
| `xcol reltime` | **igi** reads the named columns into its plot buffers. |
| `ycol roll` | Column names are not case sensitive. |
| `limits` | **igi** finds the limiting values of each buffer. |
| `erase` | Erase the graphics frame before plotting. |
| `box` | Draw box with labeled tick marks using `limits`. |
| `connect` | Connects all data points. |

The plot in figure B-1 shows a gap in the roll angle reports where the telescope was slewed away from the target, perhaps for an engineering or calibration test.

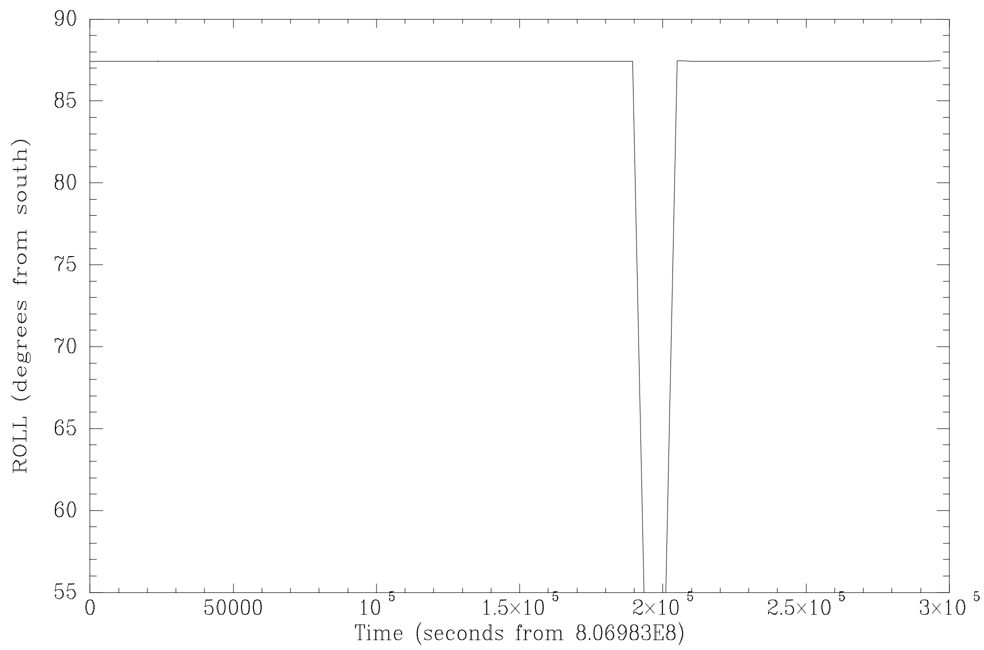


Figure B-1: ROLL vs. time - $T_0$ in **igi**. There has been a slew away from the target, as seen by the gap in the roll. The resulting large ROLL scale does not show smaller drifts of $7 - 11\widehat{''}$.

Telemetry from the test period would be omitted from that delivered to the GO[1]. The user can enter IRAF interactive cursor mode with the `curses` command and use cursor reads (`C`) to determine times just before the start and after the end of the "bad" interval.

The user can then search the times in *table1* for the expected gap, remembering that the value of $T_0$ must be added to times in the plot. It is a good idea for GO's with interrupted pointings to make sure all off-target telemetry has been omitted from the telemetry, and that any parts of the telescope slews have been masked out with a time filter on the QPOE.

Plotting the RA vs. Dec from the *mkaspectds.tab* table can also give a good idea of how stable pointing was during an observation, and the size of any drifts. Without exiting from **igi**, the user now loads the RA column of the table into `xcol`, and the Dec column into `ycol`, and recomputes the limits with the new data vectors before plotting them as in figure B-2.

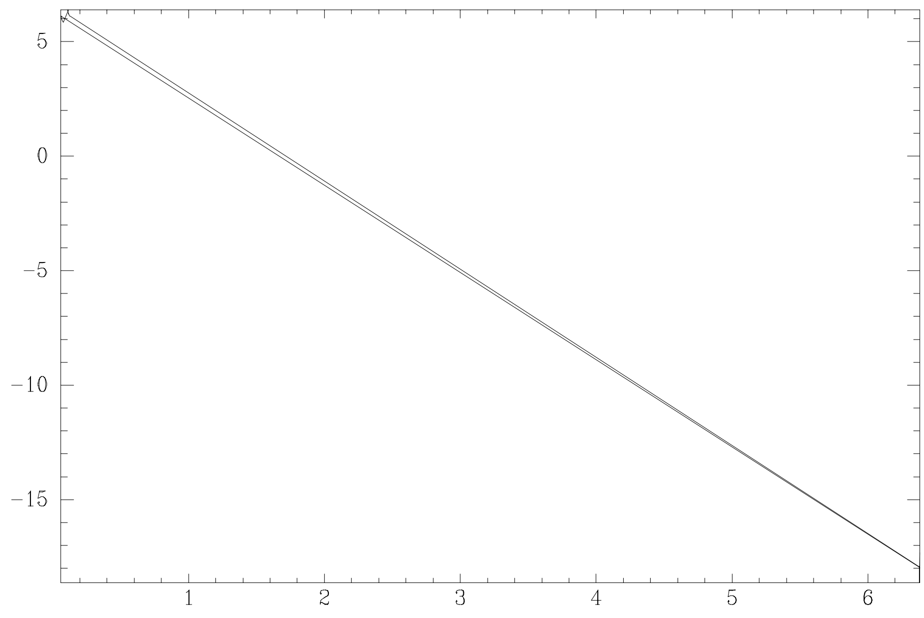


Figure B-2: RA vs. Dec for an interrupted observation. Because of the excursion, most of the pointing has been compressed into the upper left corner.

This is a "worst case" example, with the telemetry from the test included. The plot limits and area are magnified to include a location very far from the target coordinates, and the target pointings are compressed into the upper left corner of the plot. This is the kind of situation where the flexibility of **igi** is useful. The plot limits in **igi** can be reset to show the area around most of the pointings in greater

[1] Observations interrupted by tests are extended by the duration of the test to ensure the full awarded time.

detail. **igi** will use the full resolution of the graphics display to plot as many of the data values as possible in the new range, as shown in figure B-3.

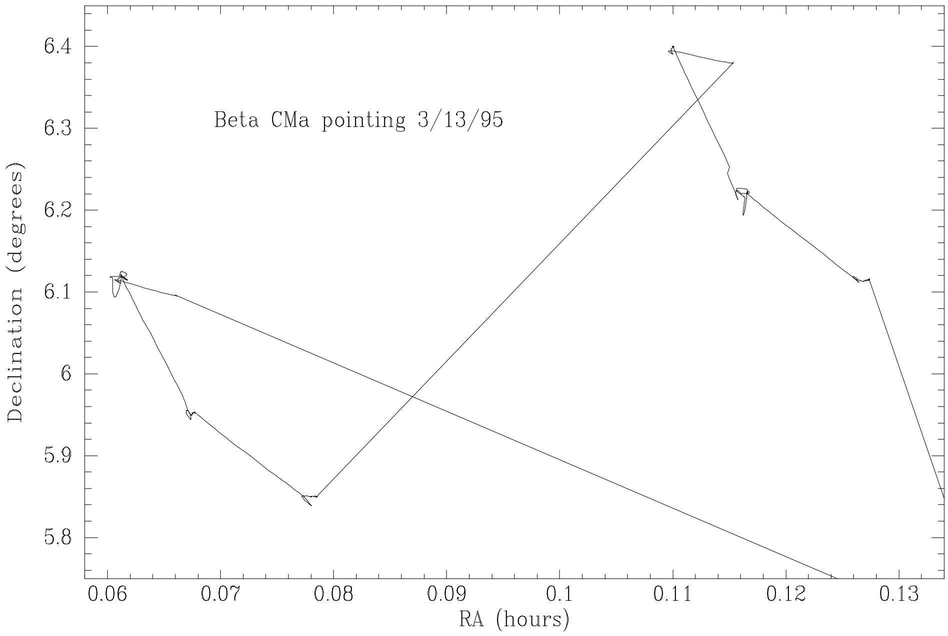


Figure B-3: Plot of RA vs. Dec for a dithered observation. The plot area is expanded about the target coordinates to show the dithering pattern of three clusters of pointing locations.

The commands for producing this plot are:

| | |
|---|---|
| `curses` | Enter interactive cursor mode |
| `C C` | Read coordinates of lower left and upper right |
| `limits .058 .134 5.75 6.45` | Close in on this area |
| `era` | Abbreviations allowed |
| `box` | Draw new coordinate axes |
| `conn` | Draw new plot |

This view shows that the observation seems to consist of several pointings, that range over about .067 hours in RA and 3/4 degree in Declination, with small clusters of corrections or dithers about each main point. Normal dithered GO observations should vary within $\pm 1 - 2'$ of the target coordinates. A hard copy of the plot with some labels can then be produced by the commands:

| | |
|---|---|
| `xlabel RA (hours)` | Label under X axis |
| `ylabel Declination (degrees)` | Label to the left of Y axis |
| `limits 0 1 0 1` | Simplifies scales for locating labels in the box |
| `relocate .15 .8` | Go to desired label location |
| `label Beta CMa pointing 3/13/95` | Label the plot |
| `curses` | Shift to IRAF cursor mode |
| `:.snap` | Send snapshot to hardcopy device |
| `q` | Return to `igi` command line mode |
| `end` | Go back to IRAF command line |

A plot of RA vs. Dec in `igi` for a typical dithered observation is shown in Chapter 1, figure 1-10.

Tips for new **igi** users

- ▽ After the `curses` command, all the regular capital letter graphics commands in the IRAF interactive cursor mode, e.g. C,X,Y,Z,M, etc., can be used to display more detail and so take advantage of the data resolution. The X-Y aspect is not preserved by the X and Y commands.
- ▽ Arguments to the label command can use a number of alternate fonts, each denoted with a letter and with a '\'. An Angstrom symbol is produced by the capital 'V' in the greek font: \gV.
- ▽ Superscripts and subscripts are also produced by \ escapes. The \u (for "up") in "cm\u2" produces $cm^2$, and the \d (for "down") in "T\d0" produces $T_0$.
- ▽ The user can edit, replay, and write the history buffer to a file:

| | |
|---|---|
| `list` | Lists previous commands, as in mongo. |
| `edit` | Puts you into the buffer in your default editor (text screen). |
| `playback` | replays the command history. |
| `write plotit` | save this buffer to file *plotit*. |

- ▽ Internal macros can be created with the `define` command:

```
igi> define limbo

macro> xcol &1
macro> ycol &2
macro> limits; erase; box
macro> end
igi> limbo table1.tab 3 5
```

creates the macro `limbo` and uses it to draw a box with labeled axes.

# OBSOLETE IRAF/EUV TASKS

The following tasks have been moved to the **euv** subpackage **obsolete**. Their use is not recommended in any current processing described in this document, but they may still be used with older EUVE data sets.

1. **listspec** is not compatible with the EUVE spectral data products in `multispec` format.
2. **euvcombine** has been replaced by **suborder** and **divea**.
3. **mkeuvspec** is no longer needed, due to improvements in the task **specmod**.
4. **editwcs** is not fully functional, and not needed for normal data processing.

In addition, the task **rowcount** is no longer available through the user interface, but is still included with the software. The following sections have not been substantially updated since the release of the Software User's Guide, version **1.4**.

## C.1 LISTSPEC Lists Spectra in WCS Units

The task **euvtools.listspec** is analagous to **images.listpixels**. It takes a 1-d spectrum as input and creates a listing of wavelength vs. the pixel values, counts or flux, according to the input. **listspec** can determine the wavelengths from the WCS in the image header, and use a bin size supplied by the user in the parameter `dx` to calculate the appropriate fluxes by linear interpolation. The user also has the option of setting the parameter `average` to divide each value by `CD1_1`, the bin width in Å. This produces an average over the bin represented by each pixel, and converts a spectrum image in photons/cm$^2$/sec to an ASCII listing in photons/cm$^2$/sec/Å. **listspec** can be used to list the output from **mkeuvspec** for comparison to the input model, or applied to flux-calibrated 1-d EUVE spectra of continuum sources to convert them to the same units as model spectra, as in the following example:

```
eu> listspec g191_all x1=70 average=yes > g191_list
```

## C.2 EUVCOMBINE Joins Channel Spectra

GO's whose spectra have very good signal-to-noise can use the task **euvcombine** to join their background-subtracted EUVE spectra before using spectral comparison tools. This task uses the instrument effective areas for the three channels to adjust the levels of overlapping spectral regions and combine the three channels, with options for fluxing the spectrum in first order and specifying the wavelength range of the output spectrum.

> **Limitations of euvcombine**
>
> Although the functionality to remove counts from orders other than first order has been implemented in **euvcombine**, this algorithm does not work reliably because of problems with noise and behavior near sharp spectral edges and features.

Therefore, **euvcombine** should only be used when the effects from higher orders can be safely ignored, as in a continuum with very little short and medium wavelength flux, or easily separated, as with an emission source that has no lines that overlap with higher order features. In these cases, the EUVE spectra can be joined using **euvcombine**, but dividing by only the first order effective area. This produces a flux-calibrated first order spectrum that can be compared directly to model spectra. GO's should look at the ST tables *sw_ea.tab*, *mw_ea.tab*, and *lw_ea.tab* in the reference data to see the relative efficiencies for higher orders.

An example parameter file setup for using **euvcombine** is shown in figure C-1.

```
eu> lpar euvcombine


     spectra = "g191sim.sw,g191sim.mw,g191sim.lw" Input spectra raw count 1d i
   detectors = "sw,mw,lw"      Input detectors for images
      output = "g191sim_com3"  Output spectra 1d image
    (refdata = "reference/detector.tab") Detector table
   (exposure = 40000.)         Exposure time in seconds
  (maxorders = 1)              Maximum number of orders
     (counts = no)             Output counts
   (autosize = no)             Determine output range from inputs
   (resample = 2)              Sampling factor to conserve memory
(minwavelengt = 70.)           Minimum output wavelength
(maxwavelengt = 760.)          Maximum output wavelength
       (mode = "al")
```

Figure C-1: Sample parameters for combining three spectra with **euvcombine**.

## Notes on running euvcombine

- Both `spectra` and `detectors` should be specified in order of increasing wavelength, and combinations of adjacent channels are allowed, as in (sw,mw,lw), (sw,mw), or (mw,lw).
- When `counts` = "no", the parameter `exposure` is used to compute photon fluxes at the telescope. In this case, the raw counts in each bin of the output spec-

trum are divided by `exposure` and the first order effective area to compute the intensity in photons/cm$^2$/sec for each pixel.

- `maxorders` was intended to determines the maximum spectral order to be included in the combined spectrum. The value of `maxorders` should always be set to 1, so that all counts are assumed to be in first order and none are subtracted (see the box above). If it is an integer greater than one, **euvcombine** attempts to remove counts from higher orders using effective areas for orders 2–`maxorder`, but it may misjudge the first order signal if it has a significant fraction of noise, and will then subtract too many counts. GO's with bright sources who ardently desire to remove higher orders are strongly urged to use the task **suborder**.
- Set `counts` to "yes" to make a raw count spectrum, "no" to make a flux calibrated spectrum.
- Set the range of the output spectrum in Å using `autosize`, to automatically select limits from the input images, or explicitly, using `minwave` and `maxwave`.
- The parameter `resample`, which specifies the binning of the output spectrum, is not currently implemented. The resolution of the output spectrum defaults to the lowest resolution present in the input spectra.
- The value of `table` should be set to the location of the ST table *detector.tab* in the directory *reference*. The table contains pointers to instrument reference data. The value of `table` should include the ".tab" extension.

To combine three channel spectra from a white dwarf observation:

```
eu> euvcombine grumpy.sw,grumpy.mw,grumpy.lw sw,mw,lw grumpy.com
exposure=39873 maxorders=1 counts-
```

To combine raw counts from the medium and long wavelength spectra of a late star:

```
eu> euvcombine latest.mw,latest.lw detectors=mw,lw counts+
```

## C.3 MKEUVSPEC Creates Images from Model Spectra

If the source model (to be input to **specmod**, version **1.4**) represents a continuum with units of photons/cm$^2$/sec/Å, or an emission source with broad or blended features, it must first be rebinned into a 1-d spectral image using the task **euvtools.mkeuvspec** to supply data in the correct units[1]. **mkeuvspec** treats an input list of fluxes as a piecewise continuous spectrum. It integrates and rebins the flux into bins with a size specified by the user.

Continuum input is simulated by a sufficiently dense array of lines, where sufficiently dense means that the line spacing is narrow compared to the spectrometer PSF that

[1] No longer true. See section 4.1.2.

will be applied to the output by **specmod** [2]. To convert a model continuum spectrum called *cont.model*, edit the parameter file for **mkeuvspec** as shown in figure C-2.

```
eu> lpar mkeuvspec


      inputs = "cont.model"     Input file list
     outputs = "cmod_in"        Output file list
       (xcol = 1)               Wavelength column
       (ycol = 2)               Y value column
         (x1 = INDEF)           First bin's wavelength
         (x2 = INDEF)           Last bin's wavelength
         (dx = 0.05)            Wavelengths per pixel
   (linelist = no)              Line list or plf
  (integrate = yes)             Is the output to be multiplied by wpc
       (mode = "al")
```

Figure C-2: Sample **mkeuvspec** parameters to make an IRAF image from an ASCII spectrum.

Notes on running **mkeuvspec**

- Letting `x1` and `x2` equal "INDEF" causes **mkeuvspec** to determine the wavelength range from the first and last values in the file.
- Setting `linelist` to "no" tells it to treat the input as a continuum and do a linear interpolation between data points.
- Setting `integrate` to "yes" causes the function to be integrated over each bin, changing the units of photons/cm$^2$/sec/Å to photons/cm$^2$/sec.
- To produce input to **specmod**, the value of the bin size in `dx` should be no more than one fourth of one resolution element in the simulated EUVE spectrum (see footnote). Values comparable to the resolution size may cause the bin size of the spectral image to "beat" against the PSF used in **specmod**, producing spurious features.

Creating simulated EUVE spectra of a continuum model in photons/cm$^2$/sec/Å from the command line requires two steps:

```
eu> mkeuvspec contin_1e18 contin18 dx=.05 linelist=no integrate=yes

eu> specmod cont18.imh contin18_sim time=50000 wavbin=.2
```

The use of **specmod** is discussed in section 4.1.2.

Making emission line lists into IRAF spectra for use as input to **specmod** is not recommended, as it will either produce inaccurate results (input bins larger) or

[2] The spectrometer resolution element is equivalent to the half width of the local PSF. Resolution elements in the short, medium and long wavelength channels are approximately .5 Å, 1.0 Å, and 2.0 Å, respectively.

slow down processing by **specmod** (input bins smaller). However, if an emission spectrum has lines or blends which already span several bins, the spectrum should be treated as a continuum by setting `linelist` to "no", and `integrate` to "yes". If emission spectra in photons/cm$^2$/sec are to be rebinned for some other purpose, set the parameter `linelist` to "yes" and `integrate` to "no". Then each line will be recorded in only one bin in the output spectrum.

# OTHER RESOURCES

## D.1 Other Useful IRAF Packages

### D.1.1 Basic image operations with generic IRAF tasks

There are a great many facilities in IRAF that you can use to move, copy, and change the one- and two- dimensional spectral images. A number of tasks in the **images** package are often useful:

```
      blkavg - Block average or sum a list of N-D images
      boxcar - Boxcar smooth a list of 1 or 2-D images
       fit1d - Fit a function to image lines or columns
       hedit - Header editor
     imarith - Simple image arithmetic
   imcombine - Combine images pixel-by-pixel using various algorithms
      imcopy - Copy an image
    imdelete - Delete a list of images
      imgets - Return the value of an image parameter as a string
    imheader - Print an image header
 imhistogram - Compute image histogram
    imrename - Rename one or more images
imstatistics - Compute and print statistics for a list of images
       imsum - Compute the sum, average, or median of a set of images
  listpixels - Convert an image section into a list of pixels
      minmax - Compute the minimum and maximum pixel values in an image
    sections - Expand an image template on the standard output
```

### D.1.2 Operating on EUVE spectra with the onedspec package

Many of the tasks in the **onedspec** package can be used on EUVE spectral images, as the WCS is compatible with the *onedspec* spectral format. Tasks include profile fitting, spectrum arithmetic, line identification, and display routines. The following is an abbreviated listing of some of the useful tasks for 1-d specta.

```
continuum - Fit the continuum in spectra
fitprofs - Fit gaussian profiles
  sarith - Spectrum arithmetic
 scombine - Combine spectra having different wavelength ranges
```

```
     scopy - Select and copy apertures in different spectral formats
     setjd - Compute and set Julian dates in images
      sfit - Fit spectra and output fit, ratio, or difference
   sinterp - Interpolate a table of x,y pairs to create a spectrum
     slist - List spectrum header parameters
  specplot - Stack and plot multiple spectra
     splot - Preliminary spectral plot/analysis
```

**splot** and **specplot** are also introduced briefly in section 1.2.1 of the *EGO Center Data Products Guide*.

### D.1.3 Manipulating QPOES with PROS XRAY tasks

The **xray** package contains a number of subpackages and many tasks that are written for use with QPOE files and the IRAF/QPOE interface. Many of the **xray** package tasks assume that the QPOE file input has a standard EINSTEIN or ROSAT event definition, making them unusable with EUVE QPOE's. However, **euv/xray** compatibility has recently been improved by the addition of the header keywords `XS-SORT` and `XS-EVENT` to EUVE QPOE files.

The tasks most often used with EUVE QPOE's are in the packages **xspatial** and **ximages**. The complete list of tasks is displayed by typing 'help xray'. Those with an asterisk may be more useful to EUVE GO's.

```
rootxray.xray:

      xapropos - List all tasks related to a given subject

      packages:

       xdataio - Tasks to convert data to/from FITS format
         xplot - Plotting routines for XRAY data
        xproto - Crude tasks, currently under development
     xobsolete - Tasks that will soon be removed from the system
*      ximages - Additions to IRAF images package

                 imcalc - IRAF image calculator
                 plcreate - create a pixel list from a region
                 pllist - display a pixel list
                 qpsort - sort a qpoe file
                 qphedit - edit header parameters in a qpoe or image file

*     xspatial - X-ray spatial analysis package

                 wcscoords - use WCS to convert between IRAF coordinate systems
                 (logical,physical)

     xspectral - X-ray spectral analysis package
```

```
*      xtiming - X-ray timing analysis package

                fft      - compute fast fourier transform on table or qpoe data
                timsort  - create a time-sorted qpoe file for timing analysis
```

The **xray** package also has several on-line help documents on various related subjects.

```
( Type help <topic> for info on any of the following: )
      pros           -     overview of all IRAF/PROS tasks

*     bugs           -  description of known IRAF/PROS bugs
      coords         -  description of IRAF coordinates used in PROS
      exposure       -     description of PROS exposure correction support
      explain_errors -  description of PROS error computations for low
                          count/bin data
      explain_xdata  -  description of new RDF file formats
      explain_screen -  description of PROS support for temporal screening
      extensions     -  description of conventions for PROS file extensions
      file_compare   -  comparison of new/RDF formats to OLD formats in PROS
*     filter         -  description of filtering options in PROS
      missions       -  list of missions and instruments support by PROS
      prf            -  information on the Point Response Function (or psf)
*     qpoe           -  description of the QPOE data file and user interface
*     regions        -  description of PROS region masks
      release_xray   -  release notes for this build
      revisions      -  revisions for the patch
```

### D.1.4 The TABLES layered packages

Several of the **tables** tasks can be useful with the EUVE telemetry tables, or to construct limit and filter specification tables. Load the **tables** package and type 'help fitsio', 'help ttools', and 'help tbplot' to produce a listing of tasks in each subpackage. Tasks which are referenced in this document and the *EGO Center Data Products Guide* are marked with an asterisk (*).

```
          ta> help fitsio



tables.fitsio:
Menu for the package tables.fitsio

    catfits - Produce a catalog of FITS files on tape or on disk
   fitscopy - Copy FITS files: disk <--> tape and disk <--> disk
fits_exampl - Detailed examples of 'fitsio' tasks
       geis - Provide background information about GEIS format files
```

```
   gftoxdim - Convert a GEIS file to one with xtra dimension plus a table
*   stwfits - Convert tables and/or GEIS files to FITS files
*   strfits - Convert FITS files to tables or to GEIS files
    r49fits - Same as strfits but handle WFPC2 data correctly.
   xdimtogf - Convert a GEIS file with xtra dimension plus a table to GEIS

(Type "help fitsio opt=sys" for more information about the fitsio package)
```

ta> help ttools

```
tables.ttools:
Menu for the package tables.ttools

       keypar - Copy an image or table header keyword to an IRAF parameter.
    keyselect - Copy selected image header keywords to sdas table.
       keytab - Copy an image or table header keyword to a table element.
       parkey - Put an IRAF parameter into an image or table header keyword.
       partab - Transfer an IRAF parameter to a table element.
       tabkey - Copy a table element to an image or table header keyword.
       tabpar - Transfer a table element to an IRAF parameter.

       gtedit - Graphically edit a table.
        gtpar - Pset to specify graph parameters for 'gtedit' task.
        imtab - Copy an image to a table column.
        tabim - Copy a table column to an image.

*       tcalc - Perform arithmetic operations on table columns.
       tchcol - Change column name, print format, or units.
*      tcheck - Check STSDAS table element values.
      tchsize - Change allocated sizes of various sections of a table.
        tcopy - Copy tables.
*     tcreate - Create a STSDAS table from an ASCII descriptor table.
      tdelete - Delete tables.
      tdiffer - Form a table which is the difference of two tables.
*       tdump - Dump the contents of a table to an ASCII file.
*       tedit - Edit a table.
      texpand - Expand tables according to a set of rules.
   thistogram - Make a histogram of a column in a table.
*       tinfo - Display table size information.
   tintegrate - Numerically integrate one column with respect to another.
        tjoin - Perform a relational join of two tables.
*       tlcol - List column information for a table.
      tlinear - Use linear regression to fit one or two table columns.
       tmatch - Find closest match between rows in two tables
       tmerge - Either merge or append tables.
       tprint - Print tables--both headers and data.
     tproduct - Form the Cartesian product of two tables.
```

```
     tproject - Create new table from selected columns in a table.
       tquery - Create a new table from selected rows and columns in a table.
*       tread - Browse through a table.
       trebin - Resample a table to uniform spacing.
      trename - Rename tables.
      tselect - Create a new table from selected rows of a table.
        tsort - Sort a table.
*       tstat - Get mean, standard deviation, min, and max for a column.
   ttranspose - Transpose or flip a table.
        tupar - Edit table header parameters.

          ta> help tbplot



tables.tbplot:
Menu for the package tables.tbplot

         axispar - Specify scaling attributes (pset for 'sgraph')
         devpar - Specify device parameters (pset for 'sgraph')
*           igi - Interactive Graphics Interpreter.
         pltpar - Specify graph parameters (pset for 'sgraph').
         sgraph - Graph one or more image sections or lists.

(Type "help tbplot option=sysdoc" for detailed information about 'tbplot').
```

The interactive graphics interpret, **igi**, is the subject of Appendix B.

## D.2 Electronic Resources for Guest Observers

The EUVE Guest Observer Center makes a number of resources available through the *ftp* server and World Wide Web (WWW) pages of the Center for EUV Astrophysics.

- The Universal Resource Locator (URL) for the EGO Center WWW page: `http://www.cea.berkeley.edu/`
  `telnet www.cea.berkeley.edu 200` For those without a WWW browser, this will start a session with the all-text browser *lynx*.
- EUVE Guest Observer Center HomePage:
  `http://www.cea.berkeley.edu/ego/HomePage.html`
- anonymous FTP:

```
ftp ftp.cea.berkeley.edu
         Name:  anonymous
         Password:  type_your_e-mail_address
```

- anonymous gopher:
  `gopher ftp.cea.berkeley.edu`
- EUVE Electronic Newsletter:
  Past issues of the EUVE electronic news are available from the CEA WWW site (URL above). Click on EUVE Public Archive of Mission Data and Information, then Documentation, then on Electronic Newsletters. Subscribe to the newsletter by sending mail to `majordomo@cea.berkeley.edu` with the one-line message body, `subscribe euvenews`.
- EUVE Guest Investigator (GI) Program:
  If you want to find out about obtaining public EUVE data, the EUVE GI Program was recently set up to help support users of the EUVE archive. For further information click on GUEST INVESTIGATOR PROGRAM in the CEA WWW Archive page, or contact the EUVE public data archive at `archive@cea.berkeley.edu`.
- Public Public Right Angle Program (RAP):
  RAP is an easy method for proposing long-exposure observations with EUVE imaging instruments. For information on the Public RAP and its streamlined proposal process, click on Public Right Angle Program Information in the EGO Program Web page Quick Reference list, or contact the EGO Center (`egoinfo@cea.berkeley.edu`). Mail all proposals to: `euverap@cea.berkeley.edu`.

## D.3 Documents You Can Get From IRAF

The following documents are among those available in the directory *iraf/docs/* in the IRAF anonymous *ftp*site. The *ftp* address is:

*iraf.noao.edu* or 140.252.1

**A Beginner's Guide to Using IRAF,**
Version 2.10, Jeannette Barnes,
August, 1993, 76 pages.
(this file is in iraf/pub/beguide.ps.Z)

**A User's Introduction to the IRAF Command Language**
Version 2.3, Peter MB Shames and Doug Tody,
August 1986, 97 pp.
(iraf/docs/cluser.ps.Z)

**The IRAF/NOAO Spectral World Coordinate Systems**
Francisco Valdez, 12 pp.
(iraf/docs/specwcs.ps.Z)

**The IRAF Packages, IRAF Version 2.10,**
6 pp.
(iraf/docs/glos210a.ps.Z)

**The NOAO Packages, IRAF Version 2.10,**
13 pp.
(iraf/docs/glos210b.ps.Z)

**Rectifying and Registering Images**
Lisa A. Wells, April 1994, 29 pp.
(iraf/docs/reg.ps.Z)

**The IRAF APEXTRACT Package (and revisions notes)**
Frank Valdez, July 1990, 11 pp.
(iraf/docs/apex.ps.Z)

**A User's Guide to Reducing Slit Spectra with IRAF**
(descriptions of **apall** and **splot**.
Phil Massey, Frank Valdes, Jeannette Barnes, 15 April 1992, 59 pp.
(iraf/docs/spect.ps.Z)

**VMS/IRAF Installation and Site Manager's Guide**
Doug Today, Suzanne Jacoby, and Nigel Sharp, July 1993, 30 pp.
(vmsiraf.ps.Z file size: 262143 uncomp.)

## D.4 Writing IRAF-Compatible Software

All these documents and many others can be gotten by anonymous ftp to *iraf.noao.edu*, in the directory *iraf/docs*. The name in parentheses is the filename under which the document is stored.

**A User's Guide to Fortran Programming in IRAF, The IMFORT Interface**
Doug Tody, September 1986, 29 pages.
(imfort.ps.Z)

**An Introductory User's Guide to IRAF Scripts,**
Ed. Anderson, revised by Rob Seaman, September 1989,, 49 pp.
(script.ps.Z)

**An Introductory User's Guide to IRAF SPP Programming,**
Rob Seaman, October 1992, 76 pages.
(sppguide.ps.Z)

## D.5 General Information on FITS

The following documents can be obtained from:

FITS Support Office
NASA/OSSA Office of Standards and Technology
Code 933
Goddard Space Flight Center
Greenbelt, MD 20771 USA

Telephone: (301) 513-1634
Internet: fits@nssdca.gsfc.nasa.gov
DECnet: NCF::FITS

Anonymous ftp site: nssdca.gsfc.nasa.gov

**A User's Guide for the Flexible Image Transport System (FITS)**
Version 3.0, October 1992.