Inter-Office Memorandum

To	Mesa Users	Date	September 26, 1980
From	Dick Sweet	Location	Palo Alto
Subject	**Mesa 6.0m Change Summary**	Organization	SDD/SS/Mesa

XEROX

Filed on: [Igor]<AlphaMesa>Doc>Summary60m.bravo

This memo outlines changes made in Mesa since the last alpha update (Mesa 6.0u, July 11, 1980). These changes will shortly be incorporated into the Mesa 6.0 Change Summary to correctly reflect the differences between Mesa 5.0 and Mesa 6.0. This document also includes a list of ARs fixed since 6.0u.

Language

The language has been extended in the following areas:

Monitor Locks and Condition Variables in Records

When constructing a record, any fields not mentioned (and not having defaults) are not guaranteed to have known values after the construction. In particular, they are not guaranteed to have their previous values. This poses a problem in types containing "hidden" fields such as the queues of processes in MONITORLOCKs and CONDITIONs, and with a few other types (see the attached memo from Ed Satterthwaite). To protect users from inadvertantly overwriting these fields, the Mesa 6 compiler does not allow updating of variables of these types, or of any composite type containing a field of these types. This does not mean that MONITORED RECORDs are read-only; they can be assigned to field by field. However, it is illegal to write a constructor for an entire MONITORED RECORD except at the point of declaration (or allocation, by the NEW operator).

Sequences

A *sequence* is an indexable collection of objects, all of which have the same type. In this respect, a sequence resembles an array; however, the length of a sequence is not specified at compile time. Consider some examples of "faked" sequences from Mesa 5.

```
. . .
ARec1: TYPE = RECORD [
    common: . . .
    count: CARDINAL, -- number of elements
    body: ARRAY [0..0) OF INTEGER];
. . .
ARec2: TYPE = RECORD [
```

```
        common: . . .
        body: ARRAY [0..0) OF INTEGER];

    ar1: POINTER TO ARec1;

    ar2: POINTER TO ARec2;
```

The record *ARec1* contains a count of the number of INTEGERs in *body*. The number of elements in an *ARec2* is not contained in the record; it is kept elsewhere, say in a global variable, or computed from the values of common fields. Note that these "variable length" arrays occur only at the end of records (they obviously couldn't occur in the middle). Also, since you aren't telling the compiler how many elements are present in a given *ARec**, it doesn't make sense to declare one; you obtain storage from your favorite allocator and access the elements using a pointer.

In Mesa 6, these constructs are built into the language. A SEQUENCE occurs as the last field of a record. It optionally has a count of elements associated with the indexable portion. The following Mesa 6 declarations are roughly equivalent to the Mesa 5 ones above:

```
. . .
SRec1: TYPE = RECORD [
    common: . . .
    body: SEQUENCE count: CARDINAL OF INTEGER];
. . .
SRec2: TYPE = RECORD [
    common: . . .
    body: SEQUENCE COMPUTED CARDINAL OF INTEGER];

sr1: POINTER TO SRec1;

sr2: POINTER TO SRec2;
```

Variables of type *SRec** must also be dynamically allocated at runtime, since they are of variable length. However, the NEW operator of Mesa 6 knows how to deal with sequences. Let *z* be an *MDSZone*, *n* a CARDINAL. The following are legal statements:

(1) *sr1* _ *z*.NEW [*SRec1*[*n*]]; -- sets *sr1.count* to *n*

(2) *sr2* _ *z*.NEW[*SRec2*[*n*]];

(3) *sr1* _ *z*.NEW[*SRec1*[*n*] _ [*common: . . ., body:* NULL]];

In statement (1), sufficient space is allocated to contain the common part of an *SRec1* plus *n* elements (INTEGERs) in the sequence part. Additionally, the number of elements, *count*, is initialized to the value *n*. Thereafter, the value of *count* is readonly. *The only way to set the length field of a sequence is by the* NEW *operator.* In statement (2), space is allocated for an *SRec2* with *n* elements, but there is no count field to initialize. Of course, in any of these statements, *n* could be replaced by any constant or expression of type CARDINAL. In statement (3), the common fields are also set by an initializing constructor. Note that the "array" part of the sequence is voided in the constructor. In Mesa 6, there is no provision for writing a constructor for the variable length portion of a SEQUENCE. One must write explicit code to initialize it.

To access an element of the sequence, one attaches an indexing expression to the pointer to the record containing the sequence. This is analogous to the treatment of type STRING (which is essentially a sequence of the Mesa 6 variety).

```
i _ sr1[j];
sr2[j] _ i;
```

If one compiles a module with bounds checking enabled (`/b` to the compiler), code will be generated when accessing *sr1*[*j*] to assure that *j* IN [0..*sr1.count*).

Alternative, on may write (analogous to saying *s.text*[*i*] for a STRING):

```
i _ sr1.body[j];
sr2.body[j] _ i;
```

The attached memo from Ed Satterthwaite discusses the use and declaration of **SEQUENCE**s in considerable detail.

Formatter

The Formatter is a program for transforming Mesa source files into a standard format. It uses the Compiler's scanner and parser to determine nesting, and hence can be used only on syntactically correct programs. Attached is the Appendix from the Mesa 6 Users Handbook (in progress) that explains the use of the Formatter.

Binder

The binder now allows the association between file names and module or configuration names to be given on the command line, in much the same way as the compiler. For example, the command line

```
>Binder.bcd Tajo[TestPilot: UnpackagedTestPilot]
```

will bind `Tajo.config` using the previously bound configuration **TestPilot** that is stored in the file named `UnpackagedTestPilot.bcd`. One can best think of the the text inside the brackets as defining (augmenting/replacing) a DIRECTORY statement at the beginning of the configuration.

System

All of the system interfaces have been recompiled for 6.0m. This means that all modules must be recompiled/rebound to use the new `Mesa.image`. No source changes are necessary (except for users of **FTPDefs**, as previously announced; see below).

To track development in other systems, minor revisions have been made in the Development Software interfaces **Format**, **Inline**, **Runtime**, **System**, and **Time**.

RunMesa

RunMesa has been upgraded to included Alto microcode for Pup checksums, IEEE floating point, and HBlt (Griffin). This microcode is loaded with the XMesa overflow microcode on Altos with the 2K ROM or 3K RAM option. Users who have been loading microcode for these functions need no longer do so. This change affects Alto IIs only.

Debugger

In addition to many bug fixes, the following changes and additions have been made:

Source Menus

`Attach`

The `Attach` menu command has been added to the *Source Ops* source menu; it causes the Debugger to ignore the creation date of the current source file when setting breakpoints or positioning to a source line. *Because this defeats type checking, this command should be used with caution*. If, after invoking this command, the Debugger sets breakpoints in strange places, chances are that the source file does not match the object in the system you are debugging.

Commands

`ReSet context [confirm]`

This command now requires two keystrokes, to avoid conflict with the `ReMote debuggee` command (not yet implemented on the Alto).

Program-not-started Indication

When the debugger refers to a program module, it usually gives the address of its global frame, e.g (`G: nnnnnB`). If the module has not been started, the debugger now prints a tilde (~) after the `B`. If a module has not been started, the user should not believe and *should not modify* the global variables of that module.

Command Central

The options window has changed format and allows inclusion of default switch settings.

Performance Monitor

The Performance Monitor has been upgraded to Mesa 6.0. It is the same as in Mesa 5.0 except that debugger breakpoint numbers are used for NodeIDs. Upgraded documentation will be forthcoming.

Xfer Counter

The Xfer Counter has been upgraded to Mesa 6.0. A new mode of operation has been added to gather information on the flow of control between groups of modules. Documentation for the Xfer Counter is attached.

Pup

PupDefs

UseAltoChecksumMicrocode has been added to speed up processing if you are running on an Alto II with the 2K ROM or 3K RAM option. The overflow microcode loaded into the RAM by `RunMesa.run` includes the necessary additions. (Beware if you load your own microcode.)

Subtle implementation changes

GetPupAddress will no longer return an address for a dying net. **EnumeratePupAddresses** will now pass the client-supplied procedure addresses on dead or dying nets, but only after processing all the addresses on nets that are reachable. (It used to skip addresses on unreachable nets.)

The byte stream internals have been reworked to eliminate several unpleasant delays while opening and closing connections. It is now possible to open a connection, send a thousand words, and close the connection in less than a second. (Since **Close** UnNews three module instances, it will take longer if you have a lot of active global frames.) A byproduct of this cleanup is that **SendNow** will send an empty **aData** packet to request an acknowledgment even if the previous **SendBlock** happened to end on a convenient packet boundary.

Bug fixes

The following change requests are closed by this release:

5005 Delays when creating byte stream
5093 **NameLookup** vs dying nets
5098 Change priority of interrupt routine in **EthernetDriver**(s)

Ftp

FTPDefs

The arguments to the (client supplied) procedure passed to **FTPInventoryDumpFile** have been extended to allow proper processing of create dates. It is now compatable with the procedure passed to **FTPEnumerateFiles**.

Bug fixes

The following change requests are closed by this release:

3987 **StringBoundsFault** from **TimeExtras.PacketTimeFromString**
4444 **FTPInventorryDumpFile** needs create date
4763 Troubles if forget to call **IdentifyNextRejectedRecipient**
5152 **TimeExtras.PacketTimeFromString** zone screwup
5198 Config with server and user things

Distribution:
Mesa Users
Mesa Group
SDSupport

Inter-Office Memorandum

To	Mesa Users	Date	September 25, 1980
From	Ed Satterthwaite	Location	Palo Alto
Subject	**Mesa 6.0m Compiler Changes**	Organization	PARC/CSL

XEROX

Filed on: [Igor]<AlphaMesa>Compiler60m.bravo (and .press)

DRAFT

This memo documents additions and changes to the Mesa 6 language since the last alpha release (Mesa 6.0u, July 11, 1980). For a complete description of the differences between the Mesa 5 and Mesa 6 languages, including the material below, see [Igor]<AlphaMesa>Compiler60.bravo (or .press).

Sequences

A *sequence* in Mesa is an indexable collection of objects, all of which have the same type. In this respect, a sequence resembles an array; however, you need not specify the length of the sequence when its type is declared, only when an instance of that type is created. Mesa 6 provides sequence-containing types for applications in which the size of a dynamically created array cannot be computed statically. Note, however, that only a subset of a more general design for sequences has been implemented. The contexts in which sequence types may appear are somewhat restricted, as are the available operations on them. We believe that the subset provides enough functionality to accomodate most uses of sequences, but you will encounter a number of annoying and sometimes inconvenient restrictions that you must take note of in your Mesa 6 programming.

One can view a sequence type as a union of some number of array types, just as the variant part of a variant record type can be viewed as a union of some (enumerated) collection of record types. Mesa adopts this view, particularly with respect to the declaration of sequence-containing types, with the following consequences:

> A sequence type can be used only to declare a field of a record. At most one such field may appear within a record, and it must occur last.
>
> A sequence-containing object has a tag field that specifies the length of that particular object and thus the set of valid indices for its elements.

To access the elements of a sequence, you use ordinary indexing operations; no discrimination is required. In this sense, all sequences are overlaid, but simple bounds checking is sufficient to validate each access.

Uses of sequence-containing variables must follow a more restrictive discipline than is currently enforced for variant records. The (maximum) length of a sequence is fixed when the object containing that sequence is created, and it cannot subsequently be changed. In addition, Mesa 6 imposes the following restrictions on the uses of sequences:

> You cannot embed a sequence-containing record within another data structure. You must allocate such records dynamically and reference them through pointers. (The NEW operation has been extended to make allocation convenient.)

You cannot derive a new type from a sequence-containing type by fixing the (maximum) length; i.e., there is no analog of a discriminated variant record type.

There are no constructors for sequence-valued components of records, nor are such components initialized automatically.

The following sections describe sequences in more detail.

Defining Sequence Types

You may use sequence types only to declare fields of records. A record may have at most one such field, and that field must be declared as the final component of the record:

Syntax

```
VariantPart          ::=   . . .
                     |     PackingOption SEQUENCE SeqTag OF TypeSpecification
SeqTag               ::=   identifier : Access  BoundsType
                     |     COMPUTED BoundsType
BoundsType           ::=   IndexType

TypeSpecification    ::=   . . .
                     |     TypeIdentifier [ Expression ]
```

The TypeSpecification in VariantPart establishes the type of the sequence elements. The BoundsType appearing in the SeqTag determines the type of the indices used to select from those elements. It is also the type of a tag value that is associated with each particular sequence object to encode the length of that object. For any such object, all valid indices are smaller than the value of the tag. If *T* is the BoundsType, the sequence type is effectively a union of array types with the index types

T[FIRST[*T*] .. FIRST[*T*]), *T*[FIRST[*T*] .. SUCC[FIRST[*T*]]), ... *T*[FIRST[*T*] .. LAST[*T*])

and a sequence with tag value *v* has index type *T*[FIRST[*T*]..*v*). Note that the smallest interval in this union is empty.

If you use the first form of SeqTag, the value of the tag is stored with the sequence and is available for subscript checking. In the form using COMPUTED, no such value is stored, and no bounds checking is possible.

Examples:

```
StackRep: TYPE = RECORD [
  top: INTEGER _ 1,
  item: SEQUENCE size: [0..LAST[INTEGER]] OF T]

Number: TYPE = RECORD [
  sign: {plus, minus},
  magnitude: SELECT kind: * FROM
    short => [val: [0..1000)],
    long => [val: LONG CARDINAL],
    extended => [val: SEQUENCE length: CARDINAL OF CARDINAL]
    ENDCASE]
```

WordSeq: TYPE = RECORD [SEQUENCE COMPUTED CARDINAL OF *Word*]

The final example illustrates the recommended method for imposing an indexable structure on raw storage.

If *S* is a type containing a sequence field, and *n* is an expression with a type conforming to CARDINAL, both *S* and *S*[*n*] are TypeSpecifications. They denote different types, however, and the valid uses of those types are different, as described below.

MACHINE DEPENDENT *Sequences*

You may declare a field with a sequence type within a MACHINE DEPENDENT record. Such a field must come last, both in the declaration and in the layout of the record, and the total length of a record with a zero-component sequence part must be a multiple of the word length. If you explicitly specify bit positions, the size of the sequence field (if given) must describe a zero-length sequence; i.e., it must account for just the space occupied by the tag field (if any). The *CharSeq* example below shows how to deal with explicit positions and computed tags.

Examples:

```
Node: TYPE = MACHINE DEPENDENT RECORD [
  info (0: 0..7): CHARACTER,
  sons (0: 8..15): SEQUENCE nSons (0: 8..15): [0..256) OF POINTER TO Node]

CharSeq: TYPE = MACHINE DEPENDENT RECORD [
  length (0): CARDINAL,
  char (1): PACKED SEQUENCE COMPUTED CARDINAL OF CHARACTER]
```

Allocating Sequences

If *S* designates a record type with a final component that is a sequence, *S*[*n*] is a type specification describing a record with a sequence part containing exactly *n* elements. The expression *n* must have a type conforming to CARDINAL. Its value need *not* be a compile-time constant; however, you can use specifications of this form only to allocate sequence-containing objects (as arguments of NEW) or to inquire about the size of such objects (as arguments of SIZE). In particular, you cannot use *S*[*n*] to define or construct a new type or to declare a variable, even for constant *n*.

The value of the expression SIZE[*S*[*n*]] has type CARDINAL and is the number of words required to store an object of type *S* having *n* components in its sequence part.

The value of the expression *z*.NEW[*S*[*n*]] has type POINTER TO *S* (or LONG POINTER TO *S* or REF *S*, depending upon the type of the zone *z*). The effect of its evaluation is to allocate SIZE[*S*[*n*]] words of storage from the zone *z* and to initialize that storage as follows:

Any fields in the common part of the record receive their default values.

The sequence tag field receives the value SUCC^{n}[FIRST[*T*]], where *T* is the type of that field.

The elements of the sequence part have undefined values.

To supply initial values for the fields in the common part, you may use a constructor for type *S* in the call of NEW. There are currently no constructors for sequence parts, however, and you must void the corresponding field. In any case, you must explicitly program any required initialization of the elements of the sequence part. In Mesa 6, this is true even if the element type has non-NULL default value.

Examples:

ps: POINTER TO *StackRep* _ z.NEW[*StackRep*[100]]; -- *s.top* = 1

pn: POINTER TO *Node* _ z.NEW[*Node*[*degree*[*c*]] _ [*info: c, sons:* NULL]]

pxn: POINTER TO *extended Number* _ *z*.NEW[*extended Number*[2**k*]]

Note that *n* specifies the maximum number of elements in the sequence part and must conform to CARDINAL no matter what BoundsType T_i appears in the SeqTag. The value assigned to the tag field is $\text{SUCC}^n[\text{FIRST}[T_i]]$. A bounds fault occurs if this is not a valid value of type T_i, i.e., if $n >$ *cardinality*(T_i), *and* you have requested bounds checking.

If $\text{FIRST}[T_i] = 0$, $\text{SUCC}^n[\text{FIRST}[T_i]]$ is just *n*, i.e., the interpretation of the tag is most intuitive if T_i is a zero-origin subrange. Usually you will specify a BoundsType (e.g., CARDINAL) with a range that comfortably exceeds the maximum expected sequence length. If, however, some maximum length *N* is important to you, you should consider using [0..*N*] as the BoundsType; then the value of the tag field in a sequence of length *n* ($n < N$) is just *n* and the valid indices are in the interval [0..*n*).

Operations on Sequences

You can use a sequence-containing type *S* only as the argument of the type constructor POINTER TO (or REF). Note that the type of *z*.NEW[*S*[*n*]] is POINTER TO *S* (not POINTER TO *S*[*n*]). If the type of an object is *S*, the operations defined upon that object are

ordinary access to fields in the common part

readonly access to the tag field (if not COMPUTED)

indexing of the sequence field

constructing a descriptor for the components of the sequence field (if not COMPUTED).

There are no other operations upon either type *S* or the sequence type embedded within *S*. In particular, you cannot assign or compare sequences or sequence-containing records (except by explicitly programming operations on the components).

Indexing: You may use indexing to select elements of the sequence-containing field of a record by using ordinary subscript notation, e.g., *s.seq*[*i*]. The type of the indexing expression *i* must conform to the BoundsType appearing in the declaration of the sequence field and must be less than the value of the tag, as described above. The result designates a variable with the type of the sequence's elements. A bounds fault occurs if the index is out of range, the sequence is not COMPUTED, *and* you have requested bounds checking.

By convention, the indexing operation upon sequences extends to records containing sequence-valued fields. Thus you need not supply the field name in the indexing operation. Note too that both indexing and field selection provide automatic dereferencing.

Examples:

ps^.*item*[*ps.top*] *ps.item*[*ps.top*] *ps*[*ps.top*] -- all equivalent

Descriptors: You may apply the DESCRIPTOR operator to the sequence field of a record; the result is a descriptor for the elements of that field. The resulting value has a descriptor type with index and component types and PACKED attribute equal to the corresponding attributes of the sequence type. By extension, DESCRIPTOR may be applied to a sequence-containing record to obtain a descriptor for the sequence part. The DESCRIPTOR operator does not automatically dereference its argument.

You cannot use the single-argument form of the DESCRIPTOR operator if the sequence is COMPUTED. The multiple-argument form remains available for constructing such descriptor values explicitly (and without type checking).

In any new programming, you should consider the following style recommendation: use sequence-containing types for allocation of arrays with dynamically computed size; use array descriptor types only for parameter passing. This style will become mandatory in the safe subset of Cedar Mesa.

Examples:

```
DESCRIPTOR[pn^]    DESCRIPTOR[pn.sons]    -- equivalent
```

String Bodies and TEXT

The type *StringBody* provided by previous versions of Mesa illustrates the intended properties and uses of sequences. For compatibility reasons, it has not been redefined as a sequence; the declarations of the types STRING and *StringBody* remain as follows:

```
STRING: TYPE = POINTER TO StringBody;

StringBody: TYPE = MACHINE DEPENDENT RECORD [
  length (0): CARDINAL _ 0,
  maxlength (1): --READONLY-- CARDINAL,
  text (2): PACKED ARRAY [0..0) OF CHARACTER]
```

The operations upon sequence-containing types have, however, been extended to *StringBody* so that its operational behavior is similar. In these extensions, the common part of the record consists of the field *length*, *maxlength* serves as the tag, and *text* is the collection of indexable components (packed characters). Thus *z*.NEW[*StringBody*[*n*]] creates a *StringBody* with *maxlength* = *n* and returns a STRING; if *s* is a STRING, *s*[*i*] is an indexing operation upon the text of *s*, DESCRIPTOR[*s*^] creates a DESCRIPTOR FOR PACKED ARRAY OF CHARACTER, etc.

> There are two anomalies arising from the actual declaration of *StringBody*: *s.text*[*i*] *never* uses bounds checking, and DESCRIPTOR[*s.text*] produces a descriptor for an array of length 0. Use *s*[*i*] and DESCRIPTOR[*s*^] instead.

Type TEXT

The following types, which describe a structure similar to a *StringBody* as a true sequence, are predeclared in Mesa 6. The type TEXT is primarily intended for users of Cedar, where the type REF TEXT (or REF READONLY TEXT) will replace most current uses of type STRING.

```
TEXT: TYPE = MACHINE DEPENDENT RECORD [
  length (0): [0..LAST[INTEGER]] _ 0,
  text (1): PACKED SEQUENCE maxLength (1): [0..LAST[INTEGER]] OF CHARACTER]
```

Restrictions on Assignment

The assignment operations defined upon certain types have been restricted so that variables of those types can be initialized (either explicitly or by default) when they are created but cannot subsequently be updated. A variable is considered to be created at its point of declaration or, for dynamically allocated objects, by the corresponding NEW operation. This restriction is made so that "invisible" fields such as queues of waiting processes cannot be smashed in a variable that is already in use.

In Mesa 6, the following types have restricted assignment operations:

MONITORLOCK

CONDITION

any type constructed using PORT

any type constructed using SEQUENCE

any type constructed using ARRAY in which the component type has a restricted assignment operation.

any type constructed using RECORD in which one of the field types has a restricted assignment operation.

Note that the restrictions upon assignment for a type do not impose restrictions upon assignment to component types. Thus selective updating of fields of a variable may be possible even when the entire variable cannot be updated; e.g., the *timeout* field of a CONDITION variable can be updated by ordinary assignment.

In Mesa 5, when a variable was allocated at runtime, it was necessary to call system procedures to initialize any fields of types MONITORLOCK or CONDITION. If one uses the Mesa 6 NEW operator, this initialization takes place automatically as a result of the defaulting mechanism.

Distribution:
Mesa Users
Mesa Group

Appendix B: Formatter

The Formatter transforms Mesa source files into a standard format. It establishes the horizontal and vertical spacing of the program in a way which reflects its logical structure.

This appendix describes the formatting rules and the operation of the formatter, including the run-time options and messages.

Preparing Source Files

See this section in Appendix A: Compiler. Since the formatter uses the scanner and parser of the compiler in order to determine structure, only syntactically correct programs may be formatted.

Examples

The formatter takes commands only from the command line: follow "`Formatter`" with a list of filenames, separated by spaces. Let us consider first a few simple examples:

The command

```
>Formatter ProgName
```

will read the file `ProgName.mesa`, copy its old contents to `Formatter.scratch$`, and assuming that it has no syntax errors, will produce a new, plain text, consistently formatted version of `ProgName.mesa`.

The command

```
>Formatter ProgName/-tk
```

will read the file `ProgName.mesa`, and produce a two column landscape listing of the module in the file `ProgName.press`. The program will be formatted using multiple fonts and faces.

There are numerous other options described below.

Command Line Description

The simplest form of command is just the name of a source file to be formatted. If you supply the command `sourcefile` with no period and no extension, the formatter assumes you mean `sourcefile.mesa`.

During formatting, the display is turned off and the compiler's pass-one die is displayed in the cursor.

The formatter reports the result of each command in `Formatter.log` with a message having one of the following forms (each * is replaced by an appropriate number; bracketed items appear only when relevant):

file.mesa -- source tokens: *, time: *

Formatting was successful. The source file has been rewritten. The original can be found in Formatter.scratch$. If several files are formatted in the same run, the original of only the last file will be in Formatter.scratch$.

file.mesa -- aborted, * errors [and * warnings] on *file*.errlog

Formatting was unsuccessful. The output of the formatter is undefined if syntax errors exist in the input file. The original file is undisturbed.

File error

The formatter could not find the specified file.

If any error or warning messages were issued, it brings this to your attention by putting "Type Key" into the cursor. The formatter will not return to the executive or run another subsystem until you acknowledge the message. (You can change this behavior by using switches, described below.)

Formatting rules

General Rule

As a general rule, the Formatter changes only the white space in the program. It does not insert or delete any printing characters. On the other hand, it may insert white space where there previously was none.

Spacing

Indentation is done by a combination of tabs and spaces in plain-text mode (assuming that a tab equals eight spaces), and by spaces alone in Bravo formatted mode.

The decision of where to break lines is made independently of the output mode: press file, plain text, or Bravo looks.

A logical unit will be placed on a single line if it fits.

A simple carriage return in the input file is treated as a space. The occurrence of two consecutive carriage returns (a blank line) is preserved in the output file. Three or more consecutive returns (two or more blank lines) result in two blank lines in the output file. Since all Bravo looks are discarded by the scanner, paragraph leading done with looks is not preserved.

For output files that contain fonts and faces (Press or Bravo) these additional rules apply:

Comments are set in italics.

The names of procedures are bold where they are defined.

Reserved words and predeclared identifiers are in Font 1.

Font 1 should be smaller than font 0. The fonts Helvetica 10 and Helvetica 8 work well in Bravo mode. For press files, the formatter choses Helvetica 10 and 8 for portrait listings and Helvetica 8 and 6 for landscape listings.

In general there are no spaces before or after atoms containing only special characters. Exceptions to this rule are as follows:

A space or carriage return follows (but does not precede) a comma, semicolon, or colon.

A space precedes a left square bracket when the bracket follows any of the keywords RECORD, MACHINE CODE, PROCEDURE, RETURNS, SIGNAL, PORT, and PROGRAM.

Spaces surround the left-arrow operator.

The exclamation point (enabling) and equal-greater (chooses) operators are always surrounded by spaces. This is also true for equal signs used in initialization and for asterisks used in place of variant record tags.

Some arithmetic operators, depending on their precedence, are surrounded by spaces.

The equivalent of two spaces are used for each level of indenting.

Structure

The formatter determines the indenting structure of the program by the brackets that surround the bodies of compounds. The brackets include **{}**, **()**, **[]**, BEGIN-END, DO-ENDLOOP, and FROM-ENDCASE. An attempt is made to maximize the amount of information on a page. For example, consider:

```
Record: TYPE = RECORD [
    field: Type,
    field: Type];
```

```
Record: TYPE = RECORD
    [
    field: Type,
    field: Type,
    ];
```

In both cases, the structure is clear; it is indicated by the indenting, not the placement of the brackets. The formatter generates the form on the left.

The body of each compound, assuming it does not fit on a single line, is indented one nesting level. The placement of the brackets depends on the bracket and on its prefix and its suffix. For example, a loop statement has the following possible prefixes, brackets, and suffixes:

Prefixes	Brackets	Suffixes
FOR, WHILE	DO	OPEN
UNTIL, (empty)	ENDLOOP	ENABLE

The following paragraphs contain a number of examples. They observe the following rules for the placement of opening and closing brackets:

The opening brackets **{**, **[**, FROM, and DO appear on the same line as their prefixes; BEGIN starts on a new line.

If the remainder of the statement fits on a single line (with its closing bracket), it is placed there, indented one level. Otherwise, all closing brackets except **]** and **}** appear on lines by themselves. If **}** is preceded by a semicolon, then it is also placed on a line by itself.

The statement following a THEN or ELSE is indented one level, unless it fits on the same line. THEN is on the same line as its matching IF, and ELSE is indented the same amount as IF.

```
IF bool THEN            IF bool THEN statement
    BEGIN               ELSE {body}
    body
    END
ELSE                    IF bool THEN {
    BEGIN                 statement;
    body                  statement}
    END
```

The labels of a SELECT (and its terminating ENDCASE) are indented one level, and the statements a second level, unless they fit on the same line with the label.

```
SELECT tag FROM
    case => statement;
    case =>
      long statement;
    ENDCASE
```

Each compound BEGIN-END, DO-ENDLOOP, or bracket pair is indented one level. When the rules for IF and SELECT call for indenting a statement, a BEGIN is not indented an extra level.

These rules are not exhaustive, but are intended to give the flavor of the formatter output.

Formatter Switches

Switches allow you to modify command input. A command has the general form

```
file[/s] {file2[/s] . . .}
```

where [] indicates an optional part and `s` is a sequence of switch specifications. A switch specification is a letter, identifying the switch, optionally preceded by a '`-`' or '`~`' to reverse its sense. The valid switches are

`c`	compile input file after formatting
`g`	don't close press file at end of input file
`h`	generate a press file (does *not* force `~t`)
`k`	generate a two-column landscape press file (does *not* force `~t`)
`p`	pause after formatting if there are errors
`r`	terminate formatting and run the program contained in file
`t`	overwrite input file with plain text formatted version (default)
`v`	overwrite input file with bravo looks using fonts 6 and 7
`z`	overwrite input file with bravo looks using fonts 0 and 1

Each switch has a default setting, The command `sourcefile` is equivalent to `sourcefile/~c~g~h~k~p~rt~u~x~z` if you use the standard defaults, i.e., the formatter only generates a plain text file to replace the original source. Note that the "`r`" switch changes the interpretation of `file`, which should name a subsystem.

If the assignment of switch names does not seem too mnemonic, realize that with the `/c` switch, all additional switches are passed to the compiler. For example,

```
>Formatter Foo/cj-a
```

would reformat `Foo.mesa` and then call

```
>Compiler Foo/j-a.
```

You can also change the default setting of any switch by using a global switch. Switches given with no sourcefile establish the default setting. Unless overridden or reset, that default applies to all subsequent commands. See, for example, the multiple program Press output example below.

Here is some information about the options:

`g` If a press file is being generated, it is not closed at the end of the current input file. It is expected that another file in the command list will also be generating press file output and a single press file will contain multiple input files. The name of the press file will be that of the first to which press output is being generated. If the type of press file (landscape versus portrait) changes, the first will be forced closed and another press file will be started. Be careful not to generate a press file larger than will be accepted by your printer.

`v, z` These switches cause the formatted version of the source file to contain bravo looks. The "z" switch is intended to be used on a standard Mesa Programming disk that has Helvetica 10 and Helvetica 8 as fonts 0 and 1. The "v" switch uses fonts 6 and 7 and produces output that is more convenient for including in documentation. Indenting is handled slightly differently for bravo format output files. In plain text mode, indentation is done by a combination of tabs and spaces: the font is assumed to be fixed pitch and the tab is assumed to be 8 times the width of a space. The z switch causes all indentation to be done with spaces only. For v, each level of indentation is indicated by a single tab.

Examples:

`foo`

Format `foo` using all the default switch settings (standard or established by a global switch).

`foo/-tk`

Formats `foo` into a two-column, landscape press file, leaving the original source unchanged.

`/-tkg ProgA ProgB ProgC ProgD/-g`

Produces a two-column, landscape press file `ProgA.press` that contains listing of all four programs, each starting on a new page.

The `r` (run) and `p` (pause) switches have identical semantics as in the compiler.

Formatter Failures

The message reporting a formatter failure has the following form:

```
FATAL FORMATTER ERROR, at id[index]:
   (source text)
Pass = 1, signal = s, message = m
```

Such a message indicates that the formatter has noticed some internal inconsistency. The formatter will skip the remainder of the command line if this happens. If you get such a message (or encounter other formatter problems), you should submit an change request as described in **Section 1.8**. Be sure to preserve the relevant files and to mention the octal codes identifying the signal (*s*) and message (*m*) in your change request. If you were overwriting the input file (i.e. not saying `/-t`) you can find the original contents of the file in `Formatter.scratch$`.

Inter-Office Memorandum

To	Mesa Users	Date	September 21, 1980
From	J. Sandman	Location	Palo Alto
Subject	Control Transfer Counting Tool	Organization	SDD/SS/Mesa

XEROX

Filed on: [IRIS]<Mesa>Doc>XferCounter.bravo .press

DRAFT

This tool for studying the behavior of Mesa programs counts the number of control transfers (**XFER**s) to a module and records the time spent executing in a module; it can also be used to gather information on the flow of control between groups of modules. An **XFER** is the general control transfer mechanism in Mesa. The following are all **XFER**s: procedure calls, returns from procedures, traps, and process switches.

The system is implemented as a set of commands that can be executed from the Mesa Debugger, a routine that intercepts all **XFER**s and collects statistics about them, plus a routine that intercepts conditional breakpoints for turning the **XFER** monitoring on and off. Existing Debugger commands are used to specify where **XFER** monitoring is enabled, and additional commands are provided for controlling the counting of **XFER**s and outputting the results.

This tool is intended to provide a global view of the behavior of a system. With this tool, a user can identify modules that warrant closer study with other tools such as the Performance Monitor.

Components

CountTool is the component of the tool that lives with user programs built on top of Alto/Mesa. This configuration contains one module: **Counter**. It contains the **XFER** trap handler and a breakpoint handler. **CountTool** must be loaded and started in the system it will monitor. This may be done by including **CountTool** in the client configuration whose control module imports and starts **XferCountDefs.Counter** or by executing the following command to the Alto Executive:

```
>Mesa CountTool Client
```

CountPackage is the component that lives as a tool in the Mesa Debugger. It implements the basic commands required to enable **XFER** monitoring and to output measurement results. **CountPackage** must be loaded into the Debugger before its commands can be executed. It is easiest to load it when installing the Debugger by executing the following command to the Alto Executive:

```
>XDebug CountPackage/l
```

The **CountPackage** creates a window through which all interactions with the tool take place.

Operation

There are two modes of operation, plain and matrix. Plain mode (the default) simply records the time spend in a module and the number of **XFER** to that module. Matrix mode is used to gather information on the flow of control between groups of modules. Each module is a member of one of 16 groups. A 16 by 16 matrix of counts and times is maintained by the Counter. The rows of the matrix are the groups of the source of the Xfer, the `from` group. The columns of the matrix are the groups of the destination of the Xfer, the `to` group.

In plain mode, when **XFER** monitoring is enabled and a **XFER** occurs, the trap handler calculates the time since the last **XFER** and adds that to the cumulative time for the current module. It then calculates which module is the destination of the **XFER** and makes that the current module, incrementing its count. In matrix mode, the trap handler updates the appropriate element of the matrix. The **XFER** handler then completes the **XFER**, and the user program continues.

The state of **XFER** monitoring can be controlled by two methods. The first is by setting a conditional break to be handled by the tool's breakpoint handler. The second is by calling the procedures **XferCountDefs.StartCounting** and **XferCountDefs.StopCounting**.

When the break handler intercepts a breakpoint, it checks to see if the breakpoint is conditional. If not, the break handler proceeds to the Debugger. Otherwise, the state of **XFER** monitoring is changed and program execution is resumed. A condition of zero turns **XFER** monitoring on; a condition of one toggles the state of **XFER** monitoring; a condition of two turns **XFER** monitoring off. Any other condition has no effect.

The procedures **XferCountDefs.StartCounting** and **XferCountDefs.StopCounting** provide an alternative method of enabling **XFER** monitoring. These procedures may be called from statements in the user's program, or they may be called using the Debugger's interpreter.

Since multiple processes may interact with each other, there is the concept of the *tracked process*. If this process is not NIL, only those **XFER**s that are encountered during execution of the tracked process are counted; all others are simply resumed. If the tracked process process is NIL, then all processes contribute to the accumulated data.

The **CountPackage** determines group membership by reading a file that associates group numbers with global frames. The easiest way to produce this file is to use the Debugger's `Display GlobalFrameTable` command and then edit the file `Debug.log`. Append the desired group number to the line for that module. If no group number is specified for a line, it goes in the group specified by the previous line. Modules not assigned group numbers are in group zero. For example:

```
StringsB, G:173134B, gfi:33B 1          group 1
StringsA, G:173140B, gfi:32B            group 1
StreamsC, G:173144B, gfi:31B 2          group 2
StreamsB, G:173150B, gfi:30B            group 2
StreamsA, G:173154B, gfi:27B            group 2
SegmentsB, G:173160B, gfi:26B 3         group 3
SegmentsA, G:173164B, gfi:25B           group 3
OurProcess, G:173170B, gfi:24B 4        group 4
```

```
NonResident, G:173210B, gfi:23B          group 4
Modules, G:173214B, gfi:22B              group 4
Miscellaneous, G:173220B, gfi:21B        group 4
MesaInit, G:173224B, gfi:20B 0           group 0
MesaDebug, G:173234B, gfi:17B            group 0
```

Window and Commands

Interaction with the **CountPackage** is through its window. There are three subwindows: the message subwindow, the form subwindow, and the log subwindow. Error messages and warnings are displayed in the message subwindow. Commands are invoked in the form subwindow. All output is displayed in the log subwindow and written on `Count.log`. An illustration of the window during a sample session is shown in Figure 1.

The elements of the form subwindow are explained below:

`Monitor: {off, on}`

Turns off/on the tools breakpoint handler. All conditional breakpoints will affect the state of **XFER** monitoring when the monitor is on, and will behave like normal conditional breakpoints when it is off.

`Zero Tables!`

Zeros out all counts and times.

`Condition Breaks!`

Makes all non-conditional breakpoints into conditional breakpoints by adding the condition "1" to them.

`Print Tables!`

Displays all the statistics for each module in order of increasing global frame table index (gfi) for plain mode. In matrix mode, it displays the statistics for each nonzero element of the matrix. The output format of times is `sec.msec:usec`. May be aborted by typing `^DEL`.

`Print Sorted!`

Displays all the statistics for each module in order of decreasing time or decreasing number of **XFER**s depending on the value of `Sort by`. May be aborted by typing `^DEL.` Not allowed in matrix mode.

`Sort by: {count, time}`

When set to `count`, the `Print Sorted` command displays in order of decreasing number of **XFER**s, otherwise it displays in order of decreasing time.

`Print Module!`

Displays the statistics for the module specified by `Module.` Not allowed in matrix mode.

`Module:`

Specifys the module to the `Print Module` command. It is either the module's global frame table index (gfi), its global frame address (g), or its module name (if the current configuration contains the desired module).

`Set Process!`

Tells the **Counter** to count only those **XFER**s that are executed by the specified process. An octal ProcessHandle as obtained from the Debugger's `List Processes` command is acceptable as input to this command. The default case is to track all processes.

`Process:`

Used by the `Set Process` command. It contains an octal ProcessHandle as obtained from the Debugger's `List Processes` command. If `Process` is empty, all processes are tracked.

`Mode: {plain, matrix}`

When set to `plain` (default) the Counter functions as in Mesa 6.0. When set to `matrix` the Counter records the flow from one group to another.

`Load Matrix!`

Using the current selection as a file name, reads the file to input group information.

`Show Group!`

Using the current selection as a group number, prints the names of the modules belonging to that group. May be aborted by typing `^DEL`.

Limitations

1. Execution Speed: Xfer monitoring slows down the executions of a program considerably since extra processing is done on every **XFER**. As a result, interrupt processes that are triggered by clocks will run relatively more frequently; e.g. the keyboard process being interrupted by the display.

2. Idle Loop Accounting: When no process is running, the Mesa Emulator runs in its idle loop waiting for a process to become ready. This idle time is charged to the process that was last running.

3. Time Base: The time base available on the Alto is a 26-bit counter, where the basic unit of time is 38 microseconds. Thus the counter turns over every 40 minutes, and no individual time greater than 40 minutes is meaningful. Total times are 32-bit numbers and will overflow after 340 minutes.

4. Overhead Calculation: Due to implementation restrictions and timer granularity, some of the overhead of processing an **XFER** trap is incorrectly assigned to the client program instead of the **CountTool**. As a result, times must be interpreted as only a relative measure of the time spent in a module.

5. Counter Sizes: Counts are 32-bit numbers. The maximum total count is 4,294,967,295 **XFER**s!

6. Memory Requirements: The **CountTool** requires seven pages of resident memory: two for its code and five for its frames and tables. This may affect the performance of some systems that use a

lot of memory, especially on the Alto.

7. **CountTool**'s break handler acts like a worry mode breakpoint; as a consequence, you may find you cannot `Quit` from the Debugger after your session. Use the `Kill` Debugger command instead.

Getting Started

Outlined below are the steps required for using the measurement tool.

1. obtain the bcd's for **CountTool** and **CountPackage**.

2. install the **CountPackage** in the Mesa Debugger (version 6.0).

3. start your program executing with the **CountTool** included.

4. enter the Debugger and set conditional breakpoints to enable monitoring as desired.

5. turn the break handler on by setting the `Monitor` parameter to `on`.

6. proceed with program execution.

7. return to the Debugger via control-swat or an unconditional breakpoint.

8. display results with the `Print` commands.

Sample Session

The following annotated listing of `Debug.log` and `Count.log` should give a fair idea of the use of the measurement tool. The Debugger was invoked by the Mesa Executive's `Debug` command

```
Alto/Mesa Debugger 6.0m of  5-Sep-80 12:02
25-Sep-80  9:56

You called?
>SEt Root configuration: MesaExec
>SEt Module context: MesaExec
>Break Entry procedure: LoadNew Breakpoint #1. -- Count XFER involved with
loading
>Break Xit procedure: LoadNew Breakpoint #2.
-- the Condition Breaks command will make these conditional breaks
-- now interact with the CountPackage. Condition breaks and set the process
>Proceed [Confirm]
You called?
-- now look at the results
>--Test.map      -- file containing group information
-- now set mode to matrix and load group information using Load Matrix command
>Proceed [Confirm]
You called?
-- now look at the matrix
>Kill session [Confirm]


Xfer Counter 6.0 of 19-Sep-80  9:53
25-Sep-80 10:48

Track process: 3647B  -- ignore keyboard and interrupt key
Conditionalized breaks


-- Output of Print Tables command with mode = plain
Total Xfers         4,088
Total Time      1.329:842
 Gfi Frame   Module                  #Xfers  %Xfers        Time  %Time
---- ------- ------------------ ----------- ------- ----------- ------
  1B 174164B Resident                    12     .29       6:286    .47
  3B 174030B DiskIO                     869   21.25     583:996  43.91
  4B 174000B Swapper                    530   12.96      96:050   7.22
  5B 173344B MDSRegion                  538   13.16     200:367  15.06
  7B 173314B BFS                          1     .02          76    .00
 10B 173304B Directory                   80    1.95      38:900   2.92
 11B 173270B DiskKD                       2     .04         533    .04
 13B 173260B Files                      140    3.42      15:392   1.15
 15B 173254B FSP                        100    2.44      15:468   1.16
 16B 173240B LoadState                   97    2.37     105:384   7.92
 22B 173214B Modules                     96    2.34      35:471   2.66
 23B 173210B NonResident                  3     .07         457    .03
 25B 173164B SegmentsA                   74    1.81      17:945   1.34
 26B 173160B SegmentsB                   89    2.17      13:487   1.01
 27B 173154B StreamsA                    55    1.34      10:629    .79
 30B 173150B StreamsB                    44    1.07       8:115    .61
 31B 173144B StreamsC                    55    1.34      10:629    .79
 32B 173140B StringsA                    89    2.17       9:525    .71
 33B 173134B StringsB                    19     .46       3:924    .29
 35B 173124B AlFont                     176    4.30      21:107   1.58
```

```
 36B 173104B AltoLoader                  237    5.79      29:070   2.18
 40B 173100B BcdOperations               153    3.74      16:421   1.23
 45B 172720B LoaderCore                  516   12.62      74:256   5.58
 47B 172660B StreamIO                     23     .56       2:057    .15
 50B 172650B SystemDisplay                82    2.00      13:830   1.03
 54B 170274B MesaExec                      8     .19         457    .03
Ignored Xfers            98
Ignored Time        309:943
```

-- Output of `Print Sorted` command with `Sorted by = count`

```
Total Xfers         4,088
Total Time      1.329:842
 Gfi Frame   Module                  #Xfers  %Xfers        Time  %Time
---- ------- ------------------ ----------- ------- ----------- ------
  3B 174030B DiskIO                    869   21.25     583:996  43.91
  5B 173344B MDSRegion                 538   13.16     200:367  15.06
  4B 174000B Swapper                   530   12.96      96:050   7.22
 45B 172720B LoaderCore                516   12.62      74:256   5.58
 36B 173104B AltoLoader                237    5.79      29:070   2.18
 35B 173124B AlFont                    176    4.30      21:107   1.58
 40B 173100B BcdOperations             153    3.74      16:421   1.23
 13B 173260B Files                     140    3.42      15:392   1.15
 15B 173254B FSP                       100    2.44      15:468   1.16
 16B 173240B LoadState                  97    2.37     105:384   7.92
 22B 173214B Modules                    96    2.34      35:471   2.66
 26B 173160B SegmentsB                  89    2.17      13:487   1.01
 32B 173140B StringsA                   89    2.17       9:525    .71
 50B 172650B SystemDisplay              82    2.00      13:830   1.03
 10B 173304B Directory                  80    1.95      38:900   2.92
 25B 173164B SegmentsA                  74    1.81      17:945   1.34
 27B 173154B StreamsA                   55    1.34      10:629    .79
 31B 173144B StreamsC                   55    1.34      10:629    .79
 30B 173150B StreamsB                   44    1.07       8:115    .61
 47B 172660B StreamIO                   23     .56       2:057    .15
 33B 173134B StringsB                   19     .46       3:924    .29
  1B 174164B Resident                   12     .29       6:286    .47
 54B 170274B MesaExec                    8     .19         457    .03
 23B 173210B NonResident                 3     .07         457    .03
 11B 173270B DiskKD                      2     .04         533    .04
  7B 173314B BFS                         1     .02          76    .00
Ignored Xfers            98
Ignored Time        309:943
```

-- Output of `Print Sorted` command with `Sorted by = time`

```
Total Xfers         4,088
Total Time      1.329:842
 Gfi Frame   Module                  #Xfers  %Xfers        Time  %Time
---- ------- ------------------ ----------- ------- ----------- ------
  3B 174030B DiskIO                    869   21.25     583:996  43.91
  5B 173344B MDSRegion                 538   13.16     200:367  15.06
 16B 173240B LoadState                  97    2.37     105:384   7.92
  4B 174000B Swapper                   530   12.96      96:050   7.22
 45B 172720B LoaderCore                516   12.62      74:256   5.58
 10B 173304B Directory                  80    1.95      38:900   2.92
 22B 173214B Modules                    96    2.34      35:471   2.66
```

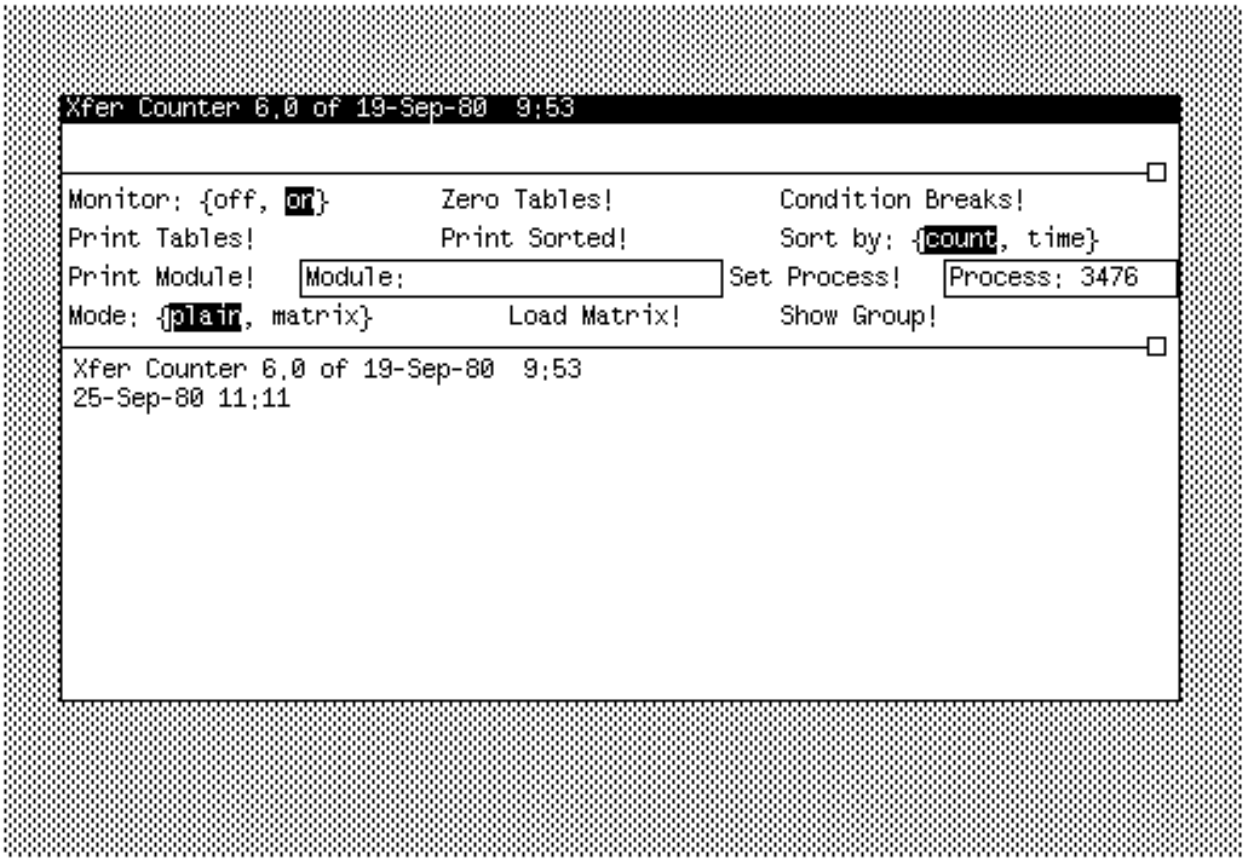
```
 36B 173104B AltoLoader                237    5.79      29:070   2.18
 35B 173124B AlFont                    176    4.30      21:107   1.58
 25B 173164B SegmentsA                  74    1.81      17:945   1.34
 40B 173100B BcdOperations             153    3.74      16:421   1.23
 15B 173254B FSP                       100    2.44      15:468   1.16
 13B 173260B Files                     140    3.42      15:392   1.15
 50B 172650B SystemDisplay              82    2.00      13:830   1.03
 26B 173160B SegmentsB                  89    2.17      13:487   1.01
 31B 173144B StreamsC                   55    1.34      10:629    .79
 27B 173154B StreamsA                   55    1.34      10:629    .79
 32B 173140B StringsA                   89    2.17       9:525    .71
 30B 173150B StreamsB                   44    1.07       8:115    .61
  1B 174164B Resident                   12     .29       6:286    .47
 33B 173134B StringsB                   19     .46       3:924    .29
 47B 172660B StreamIO                   23     .56       2:057    .15
 11B 173270B DiskKD                      2     .04         533    .04
 23B 173210B NonResident                 3     .07         457    .03
 54B 170274B MesaExec                    8     .19         457    .03
  7B 173314B BFS                         1     .02          76    .00
Ignored Xfers            98
Ignored Time        309:943


Matrix loaded

-- Output of Print Tables command with mode = matrix
Total Xfers          3,834
Total Time         871:004
From -> To           #Xfers   %Xfers            Time   %Time
----   ---     -----------  -------    -----------  ------
   1 ->  1               2      .05            114     .01
   1 ->  2               1      .02            114     .01
   1 ->  3               4      .10            266     .03
   2 ->  1               1      .02            419     .04
   2 ->  2             542    14.13        110:756   12.71
   2 ->  4             126     3.28         17:487    2.00
   2 ->  5              87     2.26         11:201    1.28
   2 ->  6               7      .18            342     .03
   2 -> 10              53     1.38         17:221    1.97
   3 ->  1               4      .10            228     .02
   3 ->  3             179     4.66         37:719    4.33
   3 ->  4              64     1.66          5:829     .66
   3 ->  6              13      .33          2:247     .25
   3 -> 10               1      .02             38     .00
   4 ->  2             124     3.23         19:240    2.20
   4 ->  3              61     1.59          6:705     .76
   4 ->  4           1,105    28.82        264:795   30.40
   4 ->  6              87     2.26         16:725    1.92
   4 ->  7               9      .23          1:790     .20
   4 ->  9               8      .20          5:486     .62
   5 ->  2              82     2.13         13:563    1.55
   6 ->  2               7      .18            495     .05
   6 ->  3              14      .36          2:057     .23
   6 ->  4              87     2.26         27:089    3.11
   6 ->  6             207     5.39         54:330    6.23
   6 ->  7              36      .93          7:658     .87
   6 -> 10              12      .31          1:066     .12
   7 ->  4               9      .23          2:552     .29
   7 ->  6              37      .96          5:067     .58
```

```
   7 ->  7             759    19.79         221:932   25.48
   9 ->  4              11      .28             647     .07
  10 ->  2              59     1.53           9:944    1.14
  10 ->  3               1      .02             152     .01
  10 ->  6              12      .31           3:543     .40
  10 -> 10              23      .59           2:171     .24

Ignored Xfers          396
Ignored Time       673:569
```


Xfer Counter 6.0 of 19-Sep-80 9:53
Monitor: {off, on}
Zero Tables!
Condition Breaks!
Print Tables!
Print Sorted!
Sort by: {count, time}
Print Module!
Module:
Set Process!
Process: 3476
Mode: {plain, matrix}
Load Matrix!
Show Group!
Xfer Counter 6.0 of 19-Sep-80 9:53
25-Sep-80 11:11

Report on file: Fixed60m.rpt1 25-Sep-80 15:54:56

Number	Originator	Subsystem	Subject
490	Murray	Binder	Compiler doesn’t check Defs for unEXPORTED exports
650	Johnsson	Binder	Packed bit on MT Records not set
801	Murray	Binder	Unbound procedure vs. Binder /r
1169	Wick	Binder	Configurations Specifying List of Control Modules
2039	Murray	Binder	Abort For ^C After Error
3128	Olmstead	Binder	Error messages re: different versions
4038	Evans	Binder	Lock released once too often when zap target bcd
4250	Knutsen	Binder	Issue Warning if two /c items
4416	Murray	Binder	Which module can’t be packed
4478	LNelson	Binder	Binder under Tajo
4502	BLyon	Binder	’IMPORT’ instead of ’IMPORTS’ in a CONFIG blew the binder up
4514	Knutsen	Binder	Copying code to bcd w/ "code" gives "Ref’d in diff versions"
4593	Luniewski	Binder	Output files incorrectly chosen
4656	mbrown	Binder	switch syntax: "bind /c foo" vs. "bind foo/c"
4658	Johnsson	Binder	Command line ’; eats next char
4676	LNelson	Binder	Binder patches for Star
4765	Newman	Binder	"Foo.run/r" puts "Foo.run/" in Com.cm
4766	Newman	Binder	"Foo.image/r" doesn’t work
4924	Newman	Binder	Fails to pause after syntax error
5392	Luniewski	Binder	Unimplemented command line args not detected
5393	Luniewski	Binder	Command line overriding of Directory statements
5727	LNelson	Binder	Fatal Binder Error: 1015B
5842	LNelson	Binder	LongBinder hung on config that standard binder handled OK
5885	Johnsson	Binder	Passing LINKS: CODE to pre-bound Config.
5887	Marzullo	Binder	binding packaged code with code copy
5950	Smokey	Binder	Fatal Binder ERROR
792	McJones	Compiler	Break Entry with DO as first statement
911	McJones	Compiler	Frames lost from UNWIND
1034	Murray	Compiler	Help on interupt routines
1969	Redell	Compiler	Bad Fine-grain Table
2068	Murray	Compiler	Poor Code For a+b*c
2622	Howard	Compiler	Fatal compiler error, Pass 5, 267601B
2858	Morrison	Compiler	Bad code generated for an INLINE (RXLPL) causes AddressFault
3335	Newman	Compiler	Doc: Pitfalls in allocating bound variants from a heap
3653	MBrown	Compiler	Error message for expressions in extractors
3660	Sweet	Compiler	bounds checking strings
3797	Howard	Compiler	RETURN WITH ERROR in ENTRY INLINE does wrong thing
4351	Guyton	Compiler	Short Pointer code generated from LONG DESCRIPTORS
4464	Tanaka.ES	Compiler	Subrange type in Defs gives bogus results
4537	McJones	Compiler	Zero-size field in M.D. record with explicit field positions
4547	Levin	Compiler	Bad FGT entry
4608	Hamilton	Compiler	ALL[ALL bombs pass 5
4657	Schwartz	Compiler	Compiler hangs in pass 5 when null EXITS clauses
4681	maxwell	Compiler	Crash in pass 5
4742	mitchell	Compiler	fatal, pass 5, variant record element compare w/nil checking
4774	ayers	Compiler	Error Messages’ Construction
4899	Sweet	Compiler	Constant Table: zero length constants entered in the table
4941	Newman	Compiler	Doc: ABORTED totally undocumented
4956	McJones	Compiler	Extra FREE for large result record in FOR loop expression
4997	Swinehart	Compiler	Machine Dependent Records Bug
5112	AWells	Compiler	Bad code generation for array initialization
5114	AWells	Compiler	Internal stack overflow in recursive type declaration
5148	Luniewski	Compiler	Fatal compiler error pass 5
5151	Murray	Compiler	FOR Clause Problem - long variable and constant bounds
5172	Nelson	Compiler	Variant record defaults
5182	Knutsen	Compiler	Comparing expr to field of result of inline blows up
5215	Newman	Compiler	Type mismatches involving zone.NEW give poor error messages
5221	AWells	Compiler	Lack of defaults for declared return variables
5231	Wyatt	Compiler	Fatal from call on an INLINE procedure
5264	LStewart	Compiler	a, b: DESCRIPTOR _ [NIL,0] => "Multiple init w/ptr"
5276	Levin	Compiler	CV and ML addresses
5281	Newman	Compiler	Confused about compile-time constants & subrange types
5294	schmidt	Compiler	NEW of extremely large arrays
5298	Artibee	Compiler	OptCatchPhrase on WaitStmt sometimes mandatory
5324	Newman	Compiler	Can’t zone.NEW for objects with non-constant size
5337	AWells	Compiler	Compiler dropping into the debugger in pass 5
5343	Levin	Compiler	@constructor gives fatal error
5385	Ludolph	Compiler	LOOP in loopexitsclause of inner loop repeats inner loop
5405	AJM@MIT-ML	Compiler	Multi-module MONITOR
5415	Daniels	Compiler	Fatal compiler error after ill-formed DIRECTORY clause
5417	Marzullo	Compiler	The same temporary location is assigned twice in a loop.
5424	Fay	Compiler	StackModelling Error in long, zero-based IN tests
5488	Knutsen	Compiler	MachDepRec complains of gaps

5492	McJones	Compiler	Catch phrase should not be allowed on RETURN WITH ERROR
5546	LStewart	Compiler	Floating point compare to zero code generator
5614	Newman	Compiler	Mainline code local variables allocated in global frame
5622	Newman	Compiler	Type mismatch for PROC RETURNS [UNCOUNTED ZONE](or MDSZone)
5687	Newman	Compiler	Bad code for record constructor containing proc calls
5689	Forrest	Compiler	Long pointer to Packed ARRAY of >4K bits generates bad code
5710	Newman	Compiler	StackModelingError, pass 5 from LONG CARDINAL loop iteration
5720	Alfvin	Compiler	NIL as an acceptable ZONE value
5754	Daniels	Compiler	Fatal System Error (Punt) in Pass 5
5783	Swinehart	Compiler	Compiler loops in pass 5 with nil-checking on
5805	Swinehart	Compiler	Code generation bug -- long pointers
5846	Sweet	Compiler	[] _ ERROR SigReturnsValues;
1161	Morrison	Debugger	Command to Redisplay Uncaught Signal Messages
2155	Karlton	Debugger	Indirect Type-in For DebugWindow
2191	Johnsson	Debugger	p%FileName$Type fails
2432	Schwartz	Debugger	Doc: Cursor actions on tiny windows need documenting
2455	Schwartz	Debugger	Doc: >> prompt not documented
2460	Schwartz	Debugger	Doc: Some finepoints
2531	malasky	Debugger	Doc: Find Variable vs Search Context
2598	Selly	Debugger	Doc: Set Octal Context examples;Frame$var
2618	CharliLevy	Debugger	Display Stack source listing in error
2767	Purvy	Debugger	Doc: Reporting debugger problems
2910	charnley	Debugger	can’t break at entry
3047	beard	Debugger	Debugger’s unawareness of new source file
3182	olmstead	Debugger	Doc: conditional breakpoints
3410	Sandman	Debugger	PerfMonitor and XferCounter trap handlers and NOOPs
3456	Murray	Debugger	-1,-1 from Trace All Xits
3610	Birrell	Debugger	Doc: Going into Swat and overwriting Swatee
3662	Birrell	Debugger	Attach Image to use current load state
3663	Birrell	Debugger	Doc: Ascii read increments start address wrong
4020	Newman	Debugger	Doc: Some variables print out followed by "^R"
4160	LNelson	Debugger	Condition breaks vs subranges
4437	Murray	Debugger	process not bound
4445	Levin	Debugger	Debugger frame trouble
4525	Freier	Debugger	Display of variable of type PROCESS
4564	Kayashima	Debugger	Xdebug Hangs when Move or Grow sized window
4598	Hamilton	Debugger	Doc: Attach Symbols
4614	Kayashima	Debugger	Stripping Bravo Trailers
4633	mbrown	Debugger	Confusion about contexts
4655	mbrown	Debugger	Attach Mesa Source command
4672	Gobbel	Debugger	Debugger bitmap goes away sometimes if Edit is used.
4680	morris	Debugger	Debugger crash
4732	birrell	Debugger	Incorrect handling of multiple instances of modules
4741	Wyatt	Debugger	COremap bug
4743	Levin	Debugger	Lockup while repainting windows
4745	Murray	Debugger	Can’t install Internal Debugger
4757	Newman	Debugger	Displaying arguments of Signaller.ErrorList goes to // mode
4759	Newman	Debugger	Fails to find symbols in files retrieved with FileTool
4761	Newman	Debugger	Misleading error message from Display Frame
4764	Newman	Debugger	Spurious "...is not a valid field selector"
4776	Murray	Debugger	1001 while trying to Kill
4777	Murray	Debugger	context mixup
4783	ayers	Debugger	Debuggers erxtra-memory bitmap space
4789	ayers	Debugger	<alphamesa>temp>xdebug.image Intall problems
4791	Murray	Debugger	Bound OVERLAID variants
4804	Levin	Debugger	Debugger bootloading: NIL PuntInfo gives PointerFault
4808	Levin	Debugger	Imported LONG DESCRIPTOR
4824	Newman	Debugger	Interpreter doesn’t know size of UNSPECIFIED
4825	Newman	Debugger	Interpeting "array[Type[value]] gives uncaught signal
4827	Newman	Debugger	Enumerated values not found
4839	birrell	Debugger	Clear All Traces missing
4843	Newman	Debugger	List Breaks says nothing if no breaks set
4844	Newman	Debugger	ATtach Conditon command rejected
4851	Cattell	Debugger	Doc: No [non-wizard] way to get debugger to redisplay SIGNAL
4858	McGregor	Debugger	Debugger Inline eval bug
4864	Cattell	Debugger	NotRelocated SIGNAL from debugger on call to user routine
4869	Johnsson	Debugger	CL? doesn’t list all options
4873	Johnsson	Debugger	? for unknown PROCEDURE
4874	Karlton	Debugger	Will not display Ascii.NULs
4875	Sweet	Debugger	Clear Break #
4877	Sweet	Debugger	Display Break #
4880	Malasky	Debugger	^Nvalidate cache command
4887	Newman	Debugger	Missing command "CLear Module" ??
4889	Newman	Debugger	ATtach Expression to nonexistent break => no error msg
4890	Newman	Debugger	LIst Breaks doesn’t display attached expressions...
4894	MBrown	Debugger	breakpoint set on "BEGIN" of proc body smashes entry break?
4916	Newman	Debugger	"Break ?" shows invalid options
4918	Newman	Debugger	"CLear ?" leaves some options out

4950	Hayes	Debugger	FileTool, PupAndFTP won’t install manually in Int. Debugger
4963	Newman	Debugger	DisplayStack-source: clobbers editable window
4966	Newman	Debugger	"Trace ?" displays invalid options
4979	Frankel	Debugger	Symbol table reset infinite loop
5014	Newman	Debugger	DisplayStack + "?" leaves some options out
5027	Newman	Debugger	Initial position of windows shouldn’t be overlaid
5028	Newman	Debugger	Interpreter won’t display variables of type MDSZone
5046	Murray	Debugger	Octal Read hangs
5119	Sapsford	Debugger	ARRAY[interval] does not account for bias
5120	birrell	Debugger	Re-setting breaks
5131	MBrown	Debugger	Doc: what does output of the interpreter’s ? operator mean?
5143	kolling	Debugger	Doc: Resetting symbol table
5176	JEllis	Debugger	Resetting Symbol Table vs Find Variable
5225	MBrown	Debugger	Display Module loses track of who is in the window
5228	Murray	Debugger	AScii Read should check Ctl-DEL
5229	Murray	Debugger	Garbage PC gives last PROCEDURE
5232	kolling	Debugger	source window selections, breakpoints, etc.
5244	Newman	Debugger	Named return values vs. exit breaks
5251	Murray	Debugger	PC to PROC mixup
5284	Newman	Debugger	"size mismatch" for Display Queue of CONDITION variable
5285	Newman	Debugger	Display Queue of MONITORLOCK => "100000B is invalid Process"
5286	Newman	Debugger	Display Queue of octal number fails
5299	Newman.ES	Debugger	Interpreter: subrange assignment error
5301	Murray	Debugger	PC mixup after clearing break
5302	Murray	Debugger	47201 from Set Module Context (ambigious)
5315	Murray	Debugger	SourceWindow: context mixup
5328	Murray	Debugger	Display GlobalFrameTable
5364	JEllis	Debugger	No response to <module name>$<variable name>
5365	JEllis	Debugger	Interpreted relational expression produces Uncaught Signal
5384	Daniels	Debugger	Falling into // mode from Interpreter
5389	morris	Debugger	Breakpoint not found.
5484	Daniels	Debugger	Incorrect printing of variant records
5695	McJones	Debugger	Spurious question mark for value of large cardinal subrange
5742	Cattell	Debugger	allows attaching condition to non-existent breakpoint
5863	Israel	Debugger	Relative string
5883	Stewart	Debugger	PACKED ARRAY printing
5910	israel	Debugger	Relative array descriptors.
1823	Hamilton	Ether	FTPtool: no filstats
2092	Malasky	Ether	ChatTool: Should Attach Or Log In
2438	Schwartz	Ether	FileTool: behaves strangely when opened a second time.
2558	LNelson	Ether	FileTool: close connection logic is defective.
2766	Hamilton	Ether	FTPtool: "IFS full" funnies
2880	Clemons	Ether	FileTool: Closes connection unnecessarily
2906	israel	Ether	Pup: FTP: Slow on Dorado
3626	Murray	Ether	FTP: sending mail
3863	Murray.PA	Ether	Fetch: Make it more like FTPtool
3900	Murray	Ether	FTP: UNWIND from FTPEnumerate/retrieve
3987	Sapsford	Ether	FTP: TimeExtras.PackedTimeFromString vs StringBoundsFault
4131	Levin	Ether	FTP: FTPServers.config
4174	Hamilton	Ether	FileTool: deletion fails
4352	Karlton	Ether	FTP: FTPAltoFile.PreProcessFile does a blind ReleaseFile
4444	Schmidt.PA	Ether	FTP: FTPInventoryDumpFile needs create date parameter
4456	Birrell	Ether	Pup: FTP: Recompile packages to fix long return record bug
4499	Murray	Ether	FTP: FTPTransferFile doesn’t pass through the creation date
4544	birrell	Ether	FTP: FTPRetrieve hangs until timeout on a "no" mark
4550	Levin	Ether	FileTool: "delete" bugs
4552	Birrell	Ether	Pup: ByteStream timeout
4590	birrell	Ether	FileTool: returns slowly if the client stops an enumeration
4612	Hamilton	Ether	FileTool: shift col.2 left; move "verify"
4763	Newman	Ether	FTP: Forgetting to call IdNxtRejRecip => disaster
4809	Levin	Ether	FileTool: Local Delete: error message is misleading
4904	Levin	Ether	FileTool: fails to restore command buttons
5005	birrell	Ether	Pup: 2 sec delay creating Byte Streams
5062	kolling	Ether	Pup: DriverDefs.debugPointer
5093	Murray	Ether	Pup: NameLookup vs dying nets
5098	Murray	Ether	Pup: EthernetDriver priority
5132	MBrown	Ether	FileTool: does not set creation date properly
5152	Murray	Ether	FTP: TimeExtras: zone screwup
5195	birrell	Ether	FileTool: window size
5198	birrell	Ether	FTP: make single config available with everything
5236	Newman	Ether	FileTool: obsolete free page count
5527	Newman	Ether	FIleTool: treat empty "source" field as "*"
1230	Judd	Other	DOC: Examples
2063	Ayers	Other	Perf. Monitor: Improved Handling Of Node String Printout
2135	Malasky	Other	Two CursorProcesses In Put After NewSession
2186	McJones	Other	Lister: Code Listings To Show Global Frame Size
2242	Johnsson	Other	IncludeChecker vs. Compiler switches
2253	Otto	Other	IncludeChecker chokes on STAR

2255	Karlton	Other	IncludeChecker should specify compilation as late as possibl
2339	Wick	Other	Copyright Notice on Mesa Source Files
2392	Ladner	Other	XFER Tool Enhancements
3303	Newman	Other	IncludeChecker: bug fixes
3744	Johnsson	Other	RunMesa 34.8 lies about widebody non XMesa machines
4166	Levin	Other	RunMesa: optionally restricting memory use
4647	Luniewski	Other	Packager: Crashes if errors while processing
4650	Luniewski	Other	Packager: /l produces both .list and .map files
4651	Luniewski	Other	Packager: Fatal error: Unbound procedure
4659	Luniewski	Other	Packager: /m option - More information requested
4663	Luniewski	Other	Packager: /m option produces fatal error
4683	Luniewski	Other	Packager: Missing file does not produce error message
4707	Luniewski	Other	Packager: multiple module instances
4737	Evans	Other	Formatter: Doesn’t DO x_SELECT..1=>n,ENDCASE=>ERROR;
4863	Luniewski	Other	Packager: Can’t process compiler bcd’s
4865	Gobbel	Other	Formatter spaces too wide
4939	Levin	Other	CommandCentral selection glitch
5058	forrest	Other	Formatter adds extra blank lines
5082	BLewis	Other	Formatter: Improper treatment of "~=" operator
5084	BLewis	Other	Formatter: Improper treatment of "~(a OR b)"
5250	Gobbel	Other	Formatter should use small font for predeclared identifiers.
5311	Sapsford	Other	Formatter: Extraneous CR
5313	Mitchell	Other	Formatter: Use of tabs and spaces
5325	Nelson	Other	Formatter handles string contants wrong
5327	Morris	Other	Formatter problem
5344	Levin	Other	Formatter: comments on END procedure lines
5434	Daniels	Other	Statistics: InsufficientVM raised upon initialization
5480	Johnsson	Other	Formatter: Overlaid variant records
2007	Murray	System	Image Files Poisoned With Bad FP
2161	Murray	System	Swapper/Disk IO Difficulties
2224	Ayers	System	ERROR in MakeCheckPoint
3061	israel	System	Nil and bounds checking
3129	Olmstead	System	Request for info
3661	Birrell	System	Aborting process in ReadChar leaves keyboard monitor locked
3668	Birrell	System	XMesa Nucleus: determining memory size
4397	Murray	System	Loading garbage: Uncaught BadFile
4405	Schmidt.PA	System	Reading leader pages: don’t change read-date
4574	Newman	System	RESUMEing a PointerFault=>uncaught SIGNAL InvalidGlobalFrame
4604	Newman	System	FrameDefs.[New/Run]Config should return a GlobalFrameHandle
4847	AWells	System	Bug in turning off echos in ReadLine (IODefs)
4932	Levin	System	Bug in AltoLoader.Load
4933	Sandman	System	BcdOps.ProcessSpaces lacks *SIZE[SpaceID]
4938	Levin	System	CommandCentral: Expand @file1 @file2 fails
4958	Levin	System	Bug in Wart: InitLoadState was changed
4970	Evans	System	Debugger looses bitmap when client makes checkpoint
4985	Newman	System	Loader doesn’t link up imported POINTER TO FRAME
5003	Evans	System	MakeImage: Disk Descriptor not swapped in when needed
5057	kolling	System	Doc: Where is the stuff that used to be in bootdefs?
5065	McGregor	System	Preserving /k when coming back from a MakeCheckpoint
5094	Murray	System	MakeImage dies
5108	Karlton	System	Time: add Time.Packed to ease compatibility problems
5147	Levin	System	Nasty FSP bug...again!
5154	Murray	System	TimeConvert.PackDT off by an hour
5184	Levin	System	HyperRegion bug: rover not in bounds
5185	Levin	System	AllocVM: specific allocation too restrictive
5190	McJones	System	CommandCentral should parse the options window fields
5191	McJones	System	Uncaught signal when CommandCentral can’t find compiler
5296	Hayes	System	Memory Competition Tajo vs. Debugger
5342	schmidt	System	GetID vs. GetId in CharIO.mesa
5416	Levin	System	Debugger smashes HyperRegion
5449	Levin	System	Missing Start Traps
5479	kolling	System	where is debugnub.bcd/mesa?
5626	Levin	System	LoadConfig returns ControlModule, not GlobalFrame
5627	Levin	System	Killing off the Interrupt process