Inter-Office Memorandum

To	Mesa Users	Date	October 17, 1977
From	Jim Sandman, John Wick	Location	Palo Alto
Subject	Mesa 3.0 Change Summary	Organization	SDD/SD

XEROX

Filed on: [MAXC]<MESA-DOC>MESASUMMARY30.BRAVO

This memo outlines changes made in Mesa since the last release (May 13, 1977).

References

The following documents can be found on [MAXC]<MESA-DOC>; all files are in Bravo format. They are available in a single package as MESA30.EARS.

Mesa 3.0 Change Summary. MESASUMMARY30.BRAVO

Mesa 3.0 Compiler Update. MESACOMPILER30.BRAVO

Mesa 3.0 Binder Update. MESABINDER30.BRAVO

Mesa 3.0 System Update. MESASYSTEM30.BRAVO

Mesa 3.0 Debugger Update. MESADEBUGGER30.BRAVO

In addition, a revision of the *Mesa Language Manual* (version 3.0) is available. The contents of the MESA-DOC directory has also been updated (the *Mesa System Documentation* and *Mesa Debugger Documentation*).

Converting to the New Binder

Mesa 3.0 introduces the notion of *configurations:* a collection of pre-bound modules. In conjunction with configurations, this release implements a new scheme for binding references between modules. These extensions require modification of all Mesa source programs. Once these changes have been made, improvements of about a factor of ten in binding time will result.

In the old scheme, procedures and signals were bound individually by name (and type) after a module was loaded. In the new scheme, procedures and signals are collected together into *interfaces* (defined by DEFINITIONS modules); binding is achieved by matching interface names. Most of the binding now takes place before modules are loaded.

> *Note: the term* interface *refers to the collection of procedures and signals (and programs) declared in a* DEFINITIONS *module. The* interface name *is the same as the* DEFINITIONS *module name.*

To use this binding scheme, each PROGRAM module must include (in its module header) a list of the interfaces it uses (in the IMPORTS clause) and a list of the interfaces it implements (in the EXPORTS clause). The following steps should be performed:

1) Ensure that all externally referenced procedures and signals are declared in some DEFINITIONS module (i.e., in some interface). This is necessary only for items defined in one module and used in another; e.g. local procedures (which are normally PRIVATE) need not appear in any DEFINITIONS module.

2) Identify the PROGRAM modules which implement each procedure and signal. These modules should export the interfaces which declare the items they implement (by listing the interface names in the EXPORTS clause of the module header).

3) Each module which uses a procedure or signal in a DEFINITIONS module should import that interface (by listing its name in the IMPORTS clause of the module header). All EXTERNAL declarations should be eliminated in this step, and replaced by a reference to some imported interface.

Users should make an initial pass over their programs, performing these steps, before attempting to compile them with the new compiler. Otherwise a very large number of error messages (one for each reference to an external item) will be generated.

Example

The following example illustrates the changes which must be made to convert to the new binder. A fragment of the StreamIO module before conversion is shown below.

```
DIRECTORY
  IODefs: FROM "iodefs",
  InlineDefs: FROM "inlinedefs",
  StreamDefs: FROM "streamdefs";

DEFINITIONS FROM StreamDefs, IODefs;

StreamIO: PROGRAM [Input, Output: StreamHandle]
  IMPLEMENTING IODefs SHARING StreamDefs = PUBLIC BEGIN

-- Externals from Strings
StringToDecimal: EXTERNAL PROCEDURE [STRING] RETURNS [INTEGER];
. . . ;

ReadDecimal: PROCEDURE RETURNS [INTEGER] =
  BEGIN
  . . . ;
  RETURN [StringToDecimal[s]]
  END;

. . . ;
END.
```

As an aid to conversion, a program has been written which makes a guess as to which included DEFINITIONS modules should be imported and exported. It also lists all external and public items which cannot be found in any included interface; these items must be assigned to some interface (or eliminated) as part of the conversion process. The program can be

found on [MAXC]<MESA>IMPEXPLISTER.IMAGE; documentation is contained in the Appendix at the end of this memo.

Part of the output generated by the import/export lister for the old version of StreamIO (and old versions of the DEFINITIONS files which it includes) is as follows:

```
Import Export Lister

File: streamio

  Imports streamdefs.xm.
    ClearCurrentLine: PROCEDURE
    ClearDisplayChar: PROCEDURE
    . . .
  UnImported EXTERNALS
    StringToDecimal: PROCEDURE
    . . .
  Exports iodefs.xm.
    WriteOctal: PROCEDURE
    WriteDecimal: PROCEDURE
    WriteNumber: PROCEDURE
    . . .
  UnExported PUBLICS
    InvalidNumberBase: ERROR
```

The sections labeled "UnImported EXTERNALS" and "UnExported PUBLICS" contain those items which could not be found in any included interface. Following step one, we find that StringToDecimal is declared in StringDefs, and that InvalidNumberBase should be declared in IODefs. (Alternately, this ERROR could be eliminated or made PRIVATE.)

The "Exports" section shows that StreamIO implements the interface defined in IODefs; this interface should be added to the EXPORTS clause (step two). In step three, StringDefs is included in the DIRECTORY statement, and both StreamDefs and StringDefs are added to the IMPORTS clause. All of the EXTERNAL declarations are also removed in this step. The final version of StreamIO appears below.

```
DIRECTORY
  IODefs: FROM "iodefs",
  InlineDefs: FROM "inlinedefs",
  StreamDefs: FROM "streamdefs"
  StringDefs: FROM "stringdefs";

DEFINITIONS FROM StreamDefs, IODefs;

StreamIO: PROGRAM [Input, Output: StreamHandle]
  IMPORTS StreamDefs, StringDefs
  EXPORTS IODefs
  SHARES StreamDefs = PUBLIC BEGIN

ReadDecimal: PROCEDURE RETURNS [INTEGER] =
  BEGIN
  . . . ;
  RETURN [StringDefs.StringToDecimal[s]]
  END;
```

```
. . . ;
END.
```

Note that InlineDefs was not involved in the modifications, since StreamIO uses no procedures or signals declared there. Note also that there has been no change in the DEFINITIONS FROM statement (or any OPEN clauses); the IMPORTS and EXPORTS lists do not automatically open included modules.

Once these steps have been completed, modules can be loaded individually as before (except that the BIND command is no longer necessary). If your application consists of several modules, you will first want to bind it into a single configuration which can be loaded as a unit (it is both faster and more space efficient to do this, instead of loading modules individually). The following section contains an example of a configuration familiar to most Mesa users.

Configurations

This section shows how one Mesa package was converted to take advantage of the new binding scheme. The Window Manager is a small configuration containing four modules: WManControl, WManPosition, WManSelection, and WManWindows. In old versions of Mesa, it also included a small control program (WManLoader) which loaded, bound, and initialized its other components. The relevant sections of this module are shown below.

```
DIRECTORY
  . . . ,
  WManagerDefs: FROM "wmanagerdefs",
  WManControl: FROM "wmancontrol",
  WManPosition: FROM "wmanposition",
  WManSelection: FROM "wmanselection",
  WManWindows: FROM "wmanwindows",
  Double: FROM "double";

DEFINITIONS FROM WManagerDefs, . . . ;

WManLoader: PROGRAM = BEGIN

  control: POINTER TO FRAME[WManControl];
  position: POINTER TO FRAME[WManPosition];
  selection: POINTER TO FRAME[WManSelection];
  windows: POINTER TO FRAME[WManWindows];

  datarec: WMDataHandle; -- common shared data

  datarec ← AllocateHeapNode[SIZE[WMDataObject]];
  control ← NEW WManControl[datarec];
  selection ← NEW WManSelection[datarec];
  windows ← NEW WManWindows[datarec];
  position ← NEW WManPosition[datarec];

  BIND selection;
  BIND windows;
  BIND position;
```

```
BIND control;

START selection;
START windows;
START position;
START control;

END.
```

Instead of this small loader, we can write a *configuration description* that specifies (to the binder) the modules to be packaged together. The description is basically just a list of modules to be included; it also specifies the interfaces needed but supplied elsewhere (by the Mesa system, in this case).

```
WManager: CONFIGURATION
  IMPORTS MenuDefs, RectangleDefs, SegmentDefs,
    StreamDefs, StringDefs, SystemDefs, WindowDefs
  EXPORTS WManagerDefs
  CONTROL WManControl =
  BEGIN
  WManControl;
  WManPosition;
  WManSelection;
  WManWindows;
  END.
```

Running this description through the binder will produce a single *binary configuration description* that can be loaded (and bound to the system) by using a single NEW command. The description also designates a CONTROL module (WManControl); its global frame will be the result of the NEW operation. When it is STARTed, it is responsible for initializing all of the other modules in the configuration. The following commands load and initialize this configuration:

```
Alto/Mesa 3.0  12-OCT-77 12:39
>New Filename: wmanager -- 132404B
>Start Global frame: 132404B
>
```

When it is started, WManControl should create the common data record and pass a pointer to it to each of the other modules (as a module parameter), just as the old version of WManLoader did. To accomplish this, the control module must obtain handles on all the other modules, in order to START them. The old loader conveniently had a POINTER TO FRAME for each module which it obtained by instantiating them individually. In the new scheme, all the modules were instantiated at once, and only the frame of the control module was returned as a result.

To obtain handles on the other modules in the configuration, we can make use of a new language feature which allows declarations of PROGRAMS as well as procedures and signals to be included in DEFINITIONS modules. Such PROGRAMS are part of the interface, and are bound just as procedures and signals in the interface are. Since we already have an interface (WManagerDefs) that is shared among all components of this configuration, we just add a few lines to it.

```
WManagerDefs: DEFINITIONS = BEGIN
  . . . ;
  WMDataHandle: TYPE = POINTER TO WMDataObject;
  . . . ;
  WManSelection: PROGRAM[WMDataHandle];
  WManWindows: PROGRAM[WMDataHandle];
  WManPosition: PROGRAM[WMDataHandle];
  . . . ;
  END.
```

With this change, the control module can obtain all the necessary frame handles from the shared interface WManagerDefs. All that remains is to move the module initialization code from the old window manager loader into the main body of WManControl.

```
DIRECTORY
  . . . ,
  WManagerDefs: FROM "wmanagerdefs";

DEFINITIONS FROM WManagerDefs, . . . ;

WManControl: PROGRAM
  IMPORTS WManagerDefs, . . .
  EXPORTS WManagerDefs =
  BEGIN

  . . . ;

  datarec:  WMDataHandle;  -- common shared data

  datarec ← AllocateHeapNode[SIZE[WMDataObject]];
  START WManSelection[datarec];
  START WManWindows[datarec];
  START WManPosition[datarec];
  . . . ;

  END.
```

Note that there has been an important change in the relationship between NEW and START: parameters are now passed by the START statement. The NEW construct creates an instance of a module (or configuration) as before, but it no longer runs any of its code. This allows binding to be performed prior to loading, and allows a configuration's control module to exercise complete control over the initialization process.

These examples illustrate the major points that must be addressed in converting to Mesa 3.0. The documents listed at the beginning of this memo contain more details.

Appendix

One of the requirements imposed by the new binding scheme is that all external references must come from DEFINITIONS modules. The Import Export Lister will facilitate the generation and modification of DEFINITIONS modules, as well as the import and export lists now required in the module heading.

The Import Export Lister processes an XM file and guesses which DEFINITIONS files should be imported and exported. The output for each file contains four sections:

1. A list of procedures and signals obtained from each included DEFINITIONS module; these interfaces should be imported.

2. Procedures and signals that were declared EXTERNAL; these should be imported from some DEFINITIONS file.

3. A list of public procedures and signals which match a like declaration in some included DEFINITIONS module. Barring name conflicts, these interfaces should be exported.

4. Those procedures or signals that are public but not declared in any included DEFINITIONS file. If any of these are referenced externally, they should be included in some exported interface.

The lister takes file names either from the command line or interactively from the keyboard. The default extension is "xm". If an included DEFINITIONS file can't be found on the disk, it is reported as missing.

The program is available on [MAXC]<OLD-MESA>IMPEXPLISTER.IMAGE. This image file runs with the old version of Mesa only (not with Mesa 3.0).

Distribution:
Mesa Users
Mesa Group