Inter-Office Memorandum

To	Mesa Users	Date	May 31, 1978
From	John Wick	Location	Palo Alto
Subject	Mesa 4.0 Binder Update	Organization	SDD/SD

XEROX

Filed on: [IRIS]<MESA>DOC>BINDER40.BRAVO

This memo outlines changes made in the Mesa binder since the last release (October 17, 1977). (In addition, the list of change requests closed by Mesa 4.0 will appear as part of the Software Release Description.)

Except for the internal BCD file format, there are no known incompatabilities with the Mesa 3.0 binder. No changes to existing configuration descriptions are required; but because of the file format change, all configurations must be rebound.

If you are not concerned with the new features described in the major headings below, and you want to get on with Mesa 4.0, skip the rest of this memo for now, and come back to it later.

Code Packing

It is now possible to pack together the code for several modules into a single segment. This is useful for two reasons:

> Since the code is allocated an integral number of pages, there is some wasted space in the last page ("breakage"). If several modules are combined into a single segment, the breakage is amortized over all the modules, and there is less waste on the average.
>
> All the modules will be brought into and out of memory together, as a unit; a reference to any module in the pack will cause all the code to be brought in. Modules which are tightly coupled dynamically are good candidates for packing (for example, resident code should probably always be packed).

Of course, it is possible to "over pack" a configuration; the segments might become so large that there will never be room in memory for more than one of them at a time (this should remind you of an overlay system). *Packing is a tradeoff, and should be used with caution.*

Syntax

The segments are specified at the beginning of the configuration by giving a list of the modules which comprise each one. Any number of PACK statements may appear. The scope of the packing specification is the whole configuration, and not subconfigurations or individual module instances, because there is at most one copy of a module's code in any configuration (if all goes well).

```
ConfigDescription ::= Directory Packing Configuration .
Packing           ::= empty | PackSeries ;
PackSeries        ::= PackList | PackSeries ; PackList
PackList          ::= PACK IdList
```

Each **PackList** defines a single segment; the code for all the modules in the **IdList** will be packed into it. The identifiers in the **IdList** must refer to modules in the configuration, and not to module instances; it is the code and not the global frames that are being packed (the frames are always packed when they are allocated by the loader).

It is illegal to specify the same module in more than one **PackList**. Even though there may be multiple instances of the module (i.e., multiple global frames) in the configuration, the code is shared by all of them, and therefore can only appear in one pack.

Finally, it is perfectly fine to reach inside a previously bound configuration that is being instantiated and single out some or all of its modules for packing. Of course, you must know something about the structure of that configuration in order to do this.

Restrictions

Obviously, the PACK statements apply only if the code is being moved to the output file; otherwise, the pack lists are ignored (and no warning message is given). This allows the programmer to debug the configuration without shuffling the code from file to file, thereby saving time. When making the final version, the packing can be effected with a binder switch, without having to modify the source of the configuration description.

Once some modules have been packed together, they cannot be taken apart and repacked with other modules later on, when they are bound into some other configuration.

Fine point:

> If a previously bound configuration contains a pack, referencing any module of the pack gets the whole thing. So it is possible to pack a module and a pack together, or even to pack two packs. It is never possible to unpack a pack.

In general, code packing should be specified only to the extent that no unpacking will ever be desired. Once the packing is done, it can't be undone, unless you start over with the individual modules.

External Links

In previous Mesa systems, links to the externals referenced by a program (imported procedures, signals, errors, frames, and programs) were always stored in the module's global frame. This allows each instance of a module to be bound differently, and it allows binding to be done at runtime without modification of the module's code segment. However, it has two drawbacks:

> The links are only referenced by the module's code, and are therefore not needed when the code is swapped out. Hence, the links logically belong in the code segment.
>
> If two instances of a module are bound identically (the usual case), the links must be stored twice.

Fine Point:

> To determine the amount of space required for external links, see the compiler's typescript file. Each link occupies one word.

The Mesa 4.0 binder therefore optionally places links in the code segment. This option is enabled by constructs in the configuration language, and is further controlled by binder and loader command modifiers (switches).

Syntax

For each component of a configuration, the link location is specified using the LINKS construct defined below. The default is frame links, as in Mesa 3.0.

```
Links          ::=  empty | LINKS : CODE | LINKS : FRAME
```

A link specification can optionally be attached to each instantiation of a module, overriding the current default, so that the link location can be different for each instance.

```
CRightSide     ::=  Item Links | Item [ ] Links | Item [ IdList ] Links
```

Alternately, the link option can be specified in the configuration header. This merely changes the default option for the configuration; it will apply to all components (including nested configurations) unless it is explicitly overridden.

```
CHead          ::=  CONFIGURATION Links Imports CExports ControlClause
```

This construction works much like the PUBLIC / PRIVATE options in Mesa, and it nests in the same way. A link option attached to a configuration changes the default for all components within it, but that default can be overriden for a particular module (or nested configuration) by specifying a different link option.

Restrictions

This scheme has the consequence that, *if a module with code links has multiple instances, each instance must be bound the same*. For example, it is usually not meaningful to specify code links if the code is shared by frames residing in several different Main Data Spaces.

As with code packing, the code links option takes effect only when the code is being moved to the output file. At this point, the binder will make room for the links as it copies the code if any module sharing that code has requested code links. Again, this allows a programmer to debug without the expense of moving the code (using frame links), and then to effect the code links option with a binder switch, without changing the source of the configuration description.

Fine point:

> Once space for code links has been added to a configuration, it cannot be undone by a later binding. On the other hand, space for code links can always be added to a (previously bound) configuration, even if it did not specify code links in its description.

Using code links has one drawback: it slows down the binding and loading process, as the code must be swapped in and rewritten. The binder must make room in the code segment for the links, as described above. And because the loader resolves imports of previously loaded modules, as well as the imports of the module being loaded, it may have to swap in (and perhaps update and swapout) the code segment for every module in the system.

Fine point:

> In an experiment, it took about 2.5 seconds to load a medium size configuration (Mesa itself) with frame links (this includes a fixed overhead of about 1.5 seconds for a directory search). With code links, factoring out the fixed overhead, it was almost eight times longer (but it still took only nine seconds).

Finally, the loader will not automatically attempt to use code links, even if the space is available in the code segment. A loader switch ("l") must be used to effect this option.

Context Switching

The command line switch /R (for run) is used to specify that the Binder should run some other program rather than returning to the Alto Executive. Both ".image" and ".run" files may be specified. If there is no explicit extension, ".image" is assumed. Any switches after the R and any other text remaining in the command line after the file with the /R switch will be copied to Com.Cm for inspection by the new program.

Examples:

> "Binder SomeConfig/g Mesa/r SomeConfig" will bind SomeConfig and then run Mesa.image as if you had typed "Mesa SomeConfig".
>
> "Binder SomeConfig/g Mesa/rd OtherConfig/-s SomeConfig" will bind SomeConfig and then run Mesa.image as if you had typed "Mesa/d OtherConfig/-s SomeConfig"
>
> "Binder SomeConfig/g Ftp.run/r Store SomeConfig.bcd" will bind SomeConfig and then run Ftp.run as if you had typed "Ftp.run Store SomeConfig.bcd"

Fine points:

> The last specification before the file with the /R switch must have the /G (go) switch to indicate the end of the previous command.
>
> You can run Bravo using the /R switch, but the current version (7.1) will not correctly find switches or arguments on the command line.

Error Messages

The binder's error messages have been improved substantially. Each message includes the corresponding source line of the configuration description (if available), and more information from the data base is available for most common errors.

Fatal errors are now reported in a fashion similar to the compiler; the signal and message are given in octal, and should be included in any change request reporting a fatal binder error.

Distribution:
  Mesa Users
  Mesa Group