Inter-Office Memorandum

To	Mesa Users	Date	October 17, 1977
From	R. Johnsson, J. Wick	Location	Palo Alto
Subject	Mesa 3.0 Binder Update	Organization	SDD/SD

XEROX

Filed on: [MAXC]<MESA-DOC>MESABINDER30.BRAVO

In this release, the binder has been changed substantially; a considerable improvment in performance has been realized as a result. The bulk of the processing is now performed by the binder prior to loading; the binder therefore operates as a separate subsystem, which is quite similar in nature to the compiler (it deals with entire modules instead of individual statements).

Input to the binder consists of a set of compiled modules and a *configuration description* which specifies what modules should be included and how they are to be bound. Output of the binder is a single *binary configuration description* (BCD) in which bindings among its modules are resolved and encoded in a form that can be quickly patched up by the loader.

Preparation of input for the binder is described in other documentation accompanying this release (in the language manual and the compiler memo), as is the processing of its output by the loader (the New command). This memo outlines the operation of the binder and the various options that are available.

Running the Binder

The binder processes a configuration description (the source) and creates a BCD file (and optionally other files containing code and symbol tables). It takes commands either from the command line or interactively from the keyboard. Commands are of the form

```
source[/s file/s file/s]
```

where [] indicates optional parts. The valid switches are

```
/d - enter debugging mode
/c - copy code segments to this file
/o - give this name to the output BCD file
/s - copy symbol segments to this file
/x - copy compressed symbols to this file
/g - begin processing the precceding files
```

While debugging, the "normal" mode of operation is to not copy code or symbol segments into the BCD (the default; no switches), but to leave them in the files generated by the compiler. This saves disk space and requires the least binding time; on the other hand, loading and debugging (and code swapping) may be a bit slower, since there are many more files to be looked up and accessed.

For "distribution", code and/or symbols can be copied into the output file by using the corresponding switch on the source file name (not on the output file name). Alternately, they can be copied into different code or symbols files by giving the file name and switch following the source file name. It is a good idea to package the symbols of your subsystem into a separate file, so that they will not take up disk space when they are not in use. (Because the new binder deals only with interfaces, symbol tables are not required for binding or loading. Of course, they are required for meaningful debugging.)

There is also an option for compressing the symbol tables as they are copied. In this mode, only public symbols declared in the global frame (including procedures and signals and their parameters and results) are included. Private symbols and variables local to procedures are not copied. This option allows limited but usually adequate debugging, and will substantially reduce the size of the symbols file (typically by 50%).

Normally a command to the binder is terminated with an end-of-line. In order to specify more than one command using the command line form of input, the /g switch (for "go") may be used to replace the end-of-line. Simply add the /g switch to the last file name of each command. (This option is not available for interactive keyboard input.)

The first file name is always the source configuration description. The last occurance of a /c, /o or /s file will prevail, and extra filenames are ignored. Default extensions are "config" for source, "bcd" for output, "code" for code and "symbols" for symbols. Default output is to source.bcd. Examples:

```
foo
```

Read foo.config and write the resulting BCD on foo.bcd. This is the "normal" debugging mode since it is the fastest and requires the least disk space.

```
foo/c
```

Read foo.config, write foo.bcd. Copy all code segments into foo.bcd. Leave all symbol segments as they were in the input files. This is a possible "distribution" mode.

```
foo/cs
```

Read foo.config, write foo.bcd. Copy all code and symbol segments into foo.bcd. This is also a possible distribution mode, if debugging will be required.

```
foo/c foo/x
```

Read foo.config, write foo.bcd. Copy all code segments into foo.bcd; compress all symbol segments into foo.symbols. By packaging all of the symbols in a single file, you minimize the risk of getting an incorrect version of some symbol table.

```
foo.cd/c foo.sym/s foo.bound/o
```

Read foo.cd, write foo.bound. Copy all code segments into foo.bound and all symbol segments into foo.sym.

```
foo.cd/c foo.sym/sg bar/c
```

Read foo.cd, write foo.bcd. Copy all code segments into foo.bcd and all symbol segments into foo.sym. Then read bar.config and write bar.bcd. Copy all its code into bar.bcd.

Because of the large number of options available, it is doubly important to maintain file consistency. Appropriate version checks are included in the binder, the loader, and the debugger. Copying code into a file other than the BCD file is supported, but probably not useful.

Current Limitations

The DIRECTORY clause in a configuration description should be used *only* when the name of a module or configuration differs from the name of its file. Do not make DIRECTORY entries for interface (DEFINITIONS) files.

The output BCD file can be renamed. The code and symbols files cannot (since the BCD contains the names of these files in its internal tables).

Forcing code and symbols into the same file (other than the BCD file) is not yet implemented.

A configuration must tell the truth about what it IMPORTS and EXPORTS, i.e. *everything* imported or exported by a configuration must actually be imported or exported by a contained module or configuration.

Multiple instances of nested configurations are not yet implemented. You can get around this by binding the nested configuration in a separate step.

The Binder will occasionally get confused after several errors and give up. Try fixing the errors and starting over. There is currently no source line output with error messages.

Estimated running time: five seconds for initialization plus one second per included file (module or configuration). Add one second per module to copy code and two seconds per module to copy symbols.