NASA Contractor Report 178383

# Utilities for Master Source Code Distribution: MAX and Friends

Carlos A. Felippa

Lockheed Missiles and Space Company, Inc.
Palo Alto, California

Contract NAS1-18444

October 1988

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

# Preface

*MAX* is a program for the manipulation of Fortran *master source code* or MSC. This is a technique by which you maintain *one and only one* master copy of your important Fortran programs under a *program development operating system*, which for *MAX* is assumed to be VAX/VMS. The master copy is not intended to be directly compiled. Instead you have to pass the MSC through MAX to produce compilable instances. These instances may correspond to different code versions (for example, double precision versus single precision), different machines (for example, IBM, CDC, Cray) or different operating systems (for example VAX/VMS vs. VAX/UNIX).

The advantages of using a master source is more pronounced in complex application programs that are developed and maintained over many years and are to be transported and executed on several computer environments. The "version lag" problem that plagues many such programs is avoided by this approach.

*MAX* is complemented by several auxiliary programs that perform nonessential functions. The ensemble is collectively known as *MAX and Friends*. All of these programs, including of course *MAX*, are executed as foreign VAX/VMS commands and can therefore be easily hidden in customized VMS command procedures.

## TABLE OF CONTENTS

* Advanced topic

# TABLE OF CONTENTS (continued)

* Advanced topic

# 1
# Introduction

## §1.1 What is *MAX*?

The Master Distributor *MAX* is a utility "filter" that takes Master Source Code (MSC) as input, and produces output tailored to specific user needs. Writing programs in Master Source Code means that you work only on *one* and *only one* master version, which contains all conceivable *instances* of code you want to submit to FORTRAN compilers or assemblers.

Some of the things *MAX* can do are: split a text file, extract selective portions of a program source file, produce VAX, CDC or UNIVAC compilable output, or configure an assembly procedure file.

*MAX* is entirely coded in FORTRAN 77, and uses only formatted FORTRAN I/O with with internal (dynamic) unit assignments. On the VAX 11/780, *MAX* can be executed through the foreign command facility of VAX/VMS; the net effect is that a *MAX* invocation looks like any other VMS command.

### When Should *MAX* be Used?

*MAX* is designed to maintain *medium-scale* FORTRAN application codes (which may possibly include FORTRAN callable assembly language modules to do special things not available in FORTRAN). By medium scale I mean substantial codes that are developed and maintained over many years, but by *one person*. What I am trying to emphasize here is that *MAX* is a *personal* utility.

*MAX* is not needed for simple "throw away" code, *i.e.* the sort of code you hack out in a few days and throw away after you are done with it. At the other extreme, *MAX* is also inappropriate for very large-scale programming projects in which a more rigorous *source control* is needed; here I am thinking of a formally-organized programming team working on a big program with over 100,000 lines of code.

To summarize: *MAX* is intended for the middle ground of scientific programming: codes in the order of 1000 to 100,000 lines, which are understood and maintained by one person, and which are likely to be transported to several computers over their lifetimes.

### What Makes *MAX* Run?

Much of the power and flexibility of *MAX* comes from its ability to process *distribution keys*. Distribution keys are labels that determine which portions of the input file are to be put out, or "distributed", for downstream use. For example, distribution key `FORTRAN` is used to extract FORTRAN-compilable code lines.

There are predefined distribution keys in *MAX*, such as `FORTRAN`, `ASSEMBLY`, `VAX`, `IBM`, etc. But most distribution keys are user-definable (where by "user" I mean the code developer). In addition, *MAX* users may define *macrokeys*, which are keys that stand for a logical combination of others; e.g., short key `UC4` might stand for something like

```
UNIVAC .OR. (CDC .AND. FTN4)
```

# 2
# Running MAX

## §2.1 HOW CAN I ACCESS MAX?

### If NICE Is Installed on Your VAX ...

On VAXs on which the NICE system is installed you probably will also find a *MAX* executable on a directory which has the logical name NICE$EXE defined by the system. You may verify that this is the case by typing

```
$ SHOW LOG NICE$EXE
```

which will tell you if NICE$EXE is defined as a system-wide logical name table.

If NICE$EXE: exists you simply insert the following foreign-command definition in your LOGIN.COM file:

```
$ MAX:==$NICE$EXE:MAX.EXE
```

After doing @LOGIN you are all set to use MAX.

There is nothing magic, by the way, about the left-hand side symbol MAX. You can select anything you please, for example if you already use the symbol MAX for other things. The symbol MAX will be used throughout this document for definiteness. Another important thing: don't forget the dollar sign before NICE$EXE.

### Otherwise ...

If NICE$EXE is not defined you will have to install *MAX and Friends* on your own. Suppose that you acquire *MAX* by some unspecified means and manage to get an executable image MAX.EXE in one of your directories. For definiteness assume that that directory is, say, DSK4:[JOHNDOE.MAX]. Then in your LOGIN.COM you insert the foreign-command definition

```
$ MAX:==$DSK4:[JOHNDOE.MAX]MAX.EXE
```

and after a @LOGIN you are ready to use MAX. Again it is important not to forget the $ before the disk name.

## §2.2 PROGRAM EXECUTION

Once the definition is installed in your LOGIN.COM file, you can invoke *MAX* just like any other VAX/VMS command. All of the information may usually be placed in a single line. In the rare cases when more that one line is needed, the VAX/VMS continuation symbol (hyphen) may be used.

If essential items such as the input and output filenames are left out, *MAX* will prompt you for them; otherwise appropriate defaults are assumed.

The MAX-invocation command may be described formally as

```
$ MAX [/Qualifiers] <Inputfile >Outputfile [#Decknames] [Userkeys]
```

where components enclosed in brackets are optional. Note that the command name (MAX being assumed here); input filename and output filename are required specifications.

Except for MAX appearing first, command components may appear *in any order*. The sequence shown above is nonetheless recommended for disciplined use.

Here is how *MAX* invocations may look (for the moment don't worry too much about the statements that follow the symbol MAX; they are explained fully in §3):

```
$ MAX <CLIP.HLP
$ MAX <CLIP.VAX >/MSC
$ MAX/FOR <CLIP.VAX >.FOR
$ MAX/FOR <CLIP.VAX >.FOR #CLIM* MACRO COMPRO
$ MAX/F/CDC <CLIP.VAX >.CDC
$ MAX/F/CDC <CLIP.VAX >.CDC FTN4 MACRO GAL
```

In these and subsequent examples, the first **$** is the VAX/VMS prompt.

## §2.3 *COMMAND PACKAGING

### Abbreviations

An advantage of the one-line reference is that the symbol-definition facility of VAX/VMS may be used to abbreviate frequently used *MAX* invocations. For example:

```
XCDCLIP:==$NICE$EXE:MAX/F/CDC  <CLIP.VAX >.CDC FTN4 MACRO GAL
```

Now entering XCDCLIP as a pseudo-command has the same effect as typing the last example line shown in §2.2.

### Command Procedure References

If parameterization of certain *MAX* command components is desirable, command procedures including one or more references to it can be readily constructed. Three examples from the author's collection are given below to clarify this point.

EXAMPLE 2.1

File [FELIPPA.MAX]XFOR.COM contains the procedure

```
$ MAX/FOR <'P1' >.FOR;999
$ DOT := 'F$LOCATE(".",P1)'
$ FNAME := 'F$EXTRACT(0,DOT,P1)'
$ SET NOON
$ FOR 'FNAME'.FOR;999
$ DEL 'FNAME'.FOR;999
```

The procedure reference may be abbreviated by defining in your LOGIN.COM file

```
$ XF :==@[FELIPPA.MAX]XFOR
```

Now typing

```
$ XF    CLIP.VAX
```

extracts the FORTRAN source from CLIP.VAX into CLIP.FOR;999, compiles it into CLIP.OBJ, and deletes the intermediate .FOR file.

REMARK 2.1

If the input filename extension is omitted, *MAX* assumes .MSC (*cf.* §3.3).

REMARK 2.2

The .FOR file is deleted regardless of whether the compilation was error-free or not, because of the presence of the SET NOON statement.

EXAMPLE 2.2

File [FELIPPA.MAX]XNICE.COM contains the procedure

```
$ MAX/FOR <'P1' >.FOR;999 'P2' 'P3' 'P4' 'P5' 'P6'
$ DOT := 'F$LOCATE(".",P1)'
$ FNAME := 'F$EXTRACT(0,DOT,P1)'
$ SET NOON
$ FOR 'FNAME'.FOR;999
$ DEL 'FNAME'.FOR;999
$ IF $STATUS THEN LIB HOME:NICE 'FNAME'.OBJ
$ IF $STATUS THEN LIB/COM HOME:NICE
$ DEL 'FNAME'.OBJ.*
$ PUR HOME:
```

The abbreviated reference is

```
$ XNICE :== @[FELIPPA.MAX]XNICE
```

Now entering

```
$ XNICE   CLIP.VAX
```

performs the same compilation sequence as in the previous example, updates and compresses object library HOME:NICE, and finally gets rid of the object file. Procedure parameters P2 through P6 are available for user key or deck name specifications.

EXAMPLE 2.3

File [FELIPPA.MAX]GETDECK.COM contains the one-line procedure

```
$ MAX #'P1'  >/MSC  <'P3'
```

The abbreviated reference is

```
$ GETDECK :== @[FELIPPA.MAX]GETDECK
```

Typing

```
$ GETDECK  SKBSOL  FROM  SKYPUL.KER
```

extracts deck SKBSOL from the MSC file SKYPUL.KER, and puts it into file SKBSOL.MSC, which is created by the run. In this abbreviated reference, FROM is a mnemonic "noise" word ignored by the GETDECK procedure.

# 3 Command Components

## §3.1 COMMAND QUALIFIERS

*MAX* command qualifiers may be categorized into four types: *execution mode specifiers, target machine identifiers, option output selectors, and prompters.* These are described in detail below.

Each command qualifier must be immediately prefixed by a slash; blanks may be entered, however, before a slash. Upper or lower case input is acceptable.

### Mode Qualifiers

These specify "*MAX* execution modes". An execution mode answers the question: what is the output intended for?

| | |
|---|---|
| *none* | *Stream mode*: no particular output use is implied. For example, if *MAX* is simply used to split a file into decks. |
| `FOR` | *FORTRAN mode*: output is to be processed by a FORTRAN compiler. |
| `ASM` | *Assembly mode*: output is to be processed by an assembler. |

Both `FOR` and `ASM` may be abbreviated to the first character.

### Machine Identifiers

These qualifiers specify the target computer for downstream use of the output. They are only effective in conjunction with the `FOR` or `ASM` execution mode. In the stream mode, machine identifiers are irrelevant.

| | |
|---|---|
| *none* | VAX computer. |
| `CDC` | CDC Cyber computers. |
| `UNI` | Univac 1100 computers. |
| `IBM` | IBM 370-type computers. |
| `CRAY` | CRAY 1 computers. |
| `CY205` | CDC Cyber 205 computers. |
| `MACH=`*Machine* | Allows the specification of a machine identifier which is not one of the above. For example, `/MACHINE=APOLLO` In this case, you can choose the machine identifier. |

## Commonly Used Output Options

These qualifiers may be used to enable or disable various output options. The basic ones are:

| | |
|---|---|
| L | Asks for informative listing: distribution keylist, and final *MAX* execution statistics (lines and decks written). If omitted, *MAX* remains silent except for error diagnostics. |
| WC | Used in conjunction with FOR or ASM to force comment lines to be written to the output. Without this option, comment lines are skipped. Meaningless in stream mode. |
| WM | Used in conjunction with FOR or ASM to force MSC control lines (those that start with C=, *=, != or .  =) to be written to the output. This is not necessary in stream mode. |
| UC | Forces conversion of all case letters to upper case. Meaningless in stream mode. |
| XM | Suppresses writing of MSC lines in stream mode. It has no effect in FOR or ASM mode. |

## *Exotic Output Options

| | |
|---|---|
| TI | In FORTRAN mode, it tells *MAX* to "translate" INCLUDE statements from the VAX form to those appropriate for Univac or CDC. (For further details, see §5.5.) |
| NL | For FOR/CDC mode only: generate appropriate C$LIST control lines for locally turning-off of compiler listing. Primarily useful to suppress repetitive COMDECK output (remember that CDC does not have an INCLUDE statement, so this is the only effective way to achieve a no-list effect). |
| SIC | For FORTRAN mode only: strip inline comments from FORTRAN statements. Inline comment text is assumed to start with the substring blank-exclamation mark-blank. (To change the midcharacter to another one, use the SIC=char form described below.) **Caution:** *MAX* doesn't check for occurrence of the separator sequence in active character strings, *e.g.*, DATA or FORMAT statements; if they occur, the strings will be stripped. |
| SIC=char | Strip inline comments separated by blank-char-blank sequence sequence. For example, MAX/F/SIC=@ specifies that inline comments follow an at-sign, for Univac. |
| HDG | For FORTRAN/Univac or Assembly/Univac mode only: generate appropriate @HDG,P control lines that display the deck name. |
| WAC | For FORTRAN mode only: force *all* comment lines to be written out. |

| | |
|---|---|
| `XDH` | For FORTRAN mode only: suppresses the output of the deck-header line (§4.7). |

***Prompt Requesters**

These qualifiers request *MAX* to prompt for certain items instead of expecting command-line specifications.

| | |
|---|---|
| `D=`*n* | where $n$ is an unsigned integer (1 through 9) asks *MAX* to prompt for $n$ deck names for selective extraction. |
| `K=`*n* | where $n$ is an unsigned integer (1 through 9) asks *MAX* to prompt for $n$ user-specified distribution keys. |

## Qualifier Examples

In the following examples, only the *MAX* invocation and qualifiers are shown for brevity (input filename, output filename and distribution keys are omitted). As always, $ is the VAX/VMS prompt.

EXAMPLE 3.1

```
$ MAX
```

No qualifiers means: stream mode, everything written out.

EXAMPLE 3.2

```
$ MAX /F
```

FORTRAN mode; VAX assumed; MSC and comment lines suppressed.

EXAMPLE 3.3

```
$ MAX/F/CDC/WC/L/TI
```

FORTRAN mode; CDC computer target; MSC lines suppressed; comments written out; informative print; INCLUDE translation.

EXAMPLE 3.4

```
$ MAX /UNI /A
```

Assembly mode; Univac computer target; MSC and comment lines suppressed.

EXAMPLE 3.5

```
$ MAX/F/K=4/D=1/SIC
```

FORTRAN mode; VAX assumed; prompt for four user keys and one deck name; strip inline comments after blank-exclamation mark-blank (default character for **SIC**).

## §3.2 INPUT FILENAME

The name of the input file, immediately preceded by <. For example:

```
$ max/f <sky:solver.msc
```

The filename is terminated by a blank or a carriage return. Note that lower case input is acceptable.

This is a *mandatory* specification. If omitted, the user will be prompted for it. On responding to be prompt, enter the filename *without* the < prefix. For example:

```
$ MAX/F >O
Input filename:  SKY:SOLVER.MSC
```

Here the **Input filename** prompt message comes from *MAX*.

If the filename extension is omitted, as well as the period, .MSC is assumed. Thus in the previous two examples one may in fact use the abbreviation SKY:SOLVER for SKY:SOLVER.MSC. To specify an input file with a blank extension, end the filename with a period and nothing afterwards (except possibly a version specification).

## §3.3 OUTPUT FILENAME

The output file is specified on the command line by prefixing the filename with a >. For example:

```
$ MAX/FOR <SOLVER  >SOLVER.FOR
```

Output goes to SOLVER.FOR, which will be created in the default directory. (Note that the input file is SOLVER.MSC.)

This is a *mandatory* specification. If omitted, the user will be prompted for it. On responding to be prompt, *omit* the > prefix.

Shorthand output-file specifications are commonly used. If only the file extension, preceded by a dot, is given, the input filename is assumed. For example:

```
$ MAX/FOR <SOLVER  >.FOR
```

This is equivalent to the previous example. The extension may be also followed by a specific version number, as in

```
$ MAX/F <SOLVER  >.FOR;4
```

although this is rarely used outside of command procedures.

If the same extension is specified, as in

```
$ MAX/FOR <MATHLIB.VAX >.VAX
```

then the operating system will create a new cycle of MATHLIB.VAX to receive the output. But this can be dangerous: one slip of the fingers typing PUR and your source is gone!

A file extension preceded by a slash forces *file splitting*. In this case ***one output file will be created for each output deck***. The name of each output file is formed by appending the specified extension to the deck name. Example:

```
$ MAX <SOLVER  >/DEK
```

If SOLVER.MSC contains four decks, say A, B, C and D, four output files named A.DEK, B.DEK, C.DEK and D.DEK will be created. Note the importance of the >; without that the /DEK specification would look the same as a command qualifier.

Occasionally it is useful (for verification or educational purposes) to make MAX output *come to the terminal*. The following specification does it:

```
$ MAX <SOLVER  >0
```

*i.e.*, a zero (*not* an oh!) after the > mark. (By the way, this command also works for the input file: <0 specifies terminal input.)

ORIGINAL PAGE IS
OF POOR QUALITY

## §3.4 *DECK SPECIFICATIONS

Most MAX operations are to be performed on *all* decks contained in the input file. No explicit deck specifications are then required. (If you don't know what a "deck" is, please skip over this subsection until you do; it is explained in §4.)

Occasionally it may be desirable to *restrict* input file processing to a few decks. To give an example, suppose that the input file BIG.ONE contains 80 decks, and it is desired to split out decks SOLV1 and SOLV2 into files SOLVE1.MSC and SOLVE2.MSC for convenient editing. The following command does it:

```
$ MAX <BIG.ONE >/MSC #SOLV1 #SOLV2
```

If these happen to be the only deck names whose first 4 characters are SOLV, this can be further abbreviated to

```
$ MAX <BIG.ONE >/MSC #SOLV*
```

which takes advantage of the name-masking capabilities of *MAX*.

General rules for deck-name specifications are:

1. Deck names must be prefixed with a pound sign, and terminated by a blank or carriage return.
2. Up to 16 deck names may be specified (but only up to 9 if a prompt-request is issued.)
3. Any deck name may contain *masking characters*, which work according to VAX/VMS name-masking conventions. For example

   ```
   #BQ*
   ```
   matches all names starting with BQ,

   ```
   #FOR%%4
   ```
   matches all six-character deck names that start with FOR and end with 4, and so on.

Qualifier D=*n* may be used to request prompting for *n* deck names (*cf.* §3.1). In response to the prompt, *omit* the pound prefix.

## §3.5 *USER-SPECIFIED DISTRIBUTION KEYS

The invocation command may contain *user keys* that control the distribution process. For example:

```
$ MAX/FOR <EZGAL.VAX >.FOR PROCEDURE MACRO
```

Here PROCEDURE and MACRO are user keys. These are recognized as such by not having any special prefix.

Another way to specify user keys relies on the use of the qualifier K=*n* (see §3.1). For example if you say

```
$ MAX/FOR/K=2 <EZGAL.VAX >.FOR
```

then you will be prompted for two keys.

Up to 16 user keys may be entered. The effect of these keys on the distribution process is explained in detail in §4.

# 4
# MSC Files: Basic Features

## §4.1 GENERAL DESCRIPTION

The input to *MAX* is an MSC file. An *MSC file* is an aggregate of decks. A *deck* is a sequence of card images identified by a name.

Decks include *MSC control statements* that describe it for various purposes. There is at least one MSC control statement per deck, which specifies the deck name; but generally there are many more. An MSC control statement begins with `C=` in its first two columns, followed by a control word (an extension of this rule is given in §4.9).

*MAX* recognizes the following MSC statements:

| | |
|---|---|
| `C=DECK` | $decknam_1$ [$decknam_2$] [*decktype*] |
| `C=BLOCK` | $key_1$ [$key_2$ ... $key_n$] |
| `C=END` | [$key_1$ ... $key_n$] |
| `C=IF` | $key_1$ [$key_2$ ... $key_n$] |
| `C=ELSEIF` | $key_1$ |
| `C=ELSE` | |
| `C=*`*macrokey* | $key_1$ [`.AND.` $key_2$ ... `.AND.` $key_n$ ] |
| `C=*`*macrokey* | $key_1$ [`.OR.` $key_2$ ... `.OR.` $key_n$] |

In the above list, upper case strings shown in typewriter font denote *literals*, i.e. they must be spelled exactly as shown. Shown in italics are labels or keys selected by the deck writer. Expressions shown in brackets are optional. Everything after the control word can be written *free-field*, with blank separators.

Here are actual examples of MSC statements so you can see how they look:

```
C=DECK SKYSOL
C=DECK SKYSOL SKYSOL FORTRAN
C=BLOCK FORTRAN
C=BLOCK CDC FTN5
C=END
C=END FORTRAN
C=END *UV
C=*CUV CDC .OR. UNIVAC .OR. VAX
C=*UA UNIVAC .AND. ASCII
C=*UAV *UA .OR. VAX
```

In this section only the `DECK`, `BLOCK` and `END` control statements will be described. Learning these will get you started and will be sufficient if your programming is largely standard FORTRAN. The other MSC statements are described under the advanced features covered in §5.

## §4.2 THE DECK CONTROL STATEMENT

The C=DECK statement identifies a deck and must be the first line of each deck. Three names may follow the control word; each name may be up to 12 characters long and should consist of alphanumeric characters only.

> **Caution**: if you plan to split MSC files on the VAX, deck names should be restricted to *nine* characters, because deck names become filenames, and VAX/VMS filenames are restricted to nine characters.

The first name identifies the deck and is mandatory. The second one should be the same as the first name. The third name identifies the deck type. If explicitly given, it should be one of these:

| | |
|---|---|
| FORTRAN | for decks containing *only* FORTRAN source code |
| ASSEMBLY | for decks containing *only* assembly-language code |
| MIXED | for decks containing a mix of FORTRAN and assembly language |
| PROCEDURE | for decks containing INCLUDE text |
| DATA | for decks containing program data |

If the deck type is omitted, FORTRAN is assumed. If the second name is omitted, a repeated deck name is assumed.

> **Caution:** Deck types PROCEDURE and DATA should not be intermixed with other deck types in the same MSC file.

## §4.3 THE BLOCK AND END CONTROL STATEMENTS

The C=BLOCK and C=END statements serve as "dividers" that label internal deck sections with *active keys.* Following the C=DECK line there are no active keys. When a C=BLOCK line is encountered, its keys are appended to the active key list. When a C=END line is encountered its keys are removed from the active key list.

A C=END statement with no keys after the END word removes the active keys introduced by the *matching* C=BLOCK line.

The following example will be used in following subsections to illustrate the concept and function of active keys:

| *Deck line* | *Text* | *Active keys* |
|---|---|---|
| 1 | C=DECK SLINE SLINE FORTRAN | |
| 2 | *mst* | |
| 3 | C=BLOCK USAGE | USAGE |
| 4 | *cmt* | USAGE |
| 5 | *cmt* | USAGE |
| 6 | C=END USAGE | USAGE |
| 7 | *mst* | |
| 8 | C=BLOCK FORTRAN | FORTRAN |
| 9 | *fst* | FORTRAN |
| 10 | *fst* | FORTRAN |
| 11 | *fst* | FORTRAN |
| 12 | C=BLOCK LAB1 | FORTRAN LAB1 |
| 13 | *fst* | FORTRAN LAB1 |
| 14 | C=BLOCK LAB2 LAB3 | FORTRAN LAB1 LAB2 LAB3 |
| 15 | *fst* | FORTRAN LAB1 LAB2 LAB3 |
| 16 | C=END LAB2 | FORTRAN LAB1 LAB2 LAB3 |
| 17 | *fst* | FORTRAN LAB1 LAB3 |
| 18 | *cmt* | FORTRAN LAB1 LAB3 |
| 19 | C=END LAB1 LAB3 | FORTRAN LAB1 LAB3 |
| 20 | *fst* | FORTRAN |
| 21 | C=END FORTRAN | FORTRAN |

In the above,

| | |
|---|---|
| *fst* | FORTRAN statement |
| *cmt* | FORTRAN comment |
| *mst* | Control statement ignored by *MAX*. For example, C=AUTHOR, C=EQUIPMENT, C=VERSION, ... These statements are used primarily for documentation rather than control purposes, and they recognized by the *SCAM* utility described in §8. |

## §4.4 DISTRIBUTION RULES

Distribution rules vary according to the *MAX* execution mode. (If you don't remember what "execution modes" are, please reread §3.1.)

### Stream Mode Distribution

A deck line, including MSC lines unless the XM qualifier is on, is output if *each of its active keys is in the distribution key list, and the latter is nonempty.* An empty distribution list receives special treatment (see §4.6).

For example, suppose that the distribution key list for the sample deck above is

```
FORTRAN LAB1 LAB3
```

In stream mode without the XM qualifier, lines 8 thru 13, 17 thru 21 (inclusive) will be written out (total = 11 lines).

But if the XM qualifier is specified, only non-MSC lines 9, 10, 11, 13, 17, 18 and 20 will be written out (total = 7 lines).

### FOR or ASM Mode Distribution

In FORTRAN or Assembly mode, each *compilable source line* with active keys in the distribution list is written to output. Comment lines are not written unless the WC qualifier appears; MSC lines are not written unless the WM qualifier appears. The deck identification line is suitably transformed for compiler compatibility (*cf.* §4.7).

Going back again to the sample deck of §4.3, suppose that the distribution key list is:

```
FORTRAN LAB1 LAB2 LAB3
```

In the FORTRAN mode without WC and WM qualifiers, only the seven lines 9, 10, 11, 13, 15 and 20, plus a machine-dependent header line, will be output. If WC is specified, lines 4, 5 and 18 also are deleted. If both WC and WM are specified, lines 8 through 21 (inclusive) are deleted.

## §4.5 DISTRIBUTION KEYS

How are distribution keys specified? A distribution key may be of two types: *implied key* or *user key.*

Implied distribution keys are the result of qualifier options. The following table summarizes all possibilities.

| *Mode* | *Machine* | *Implied keys* |
|---|---|---|
| Stream | irrelevant | none |
| FORTRAN | VAX (default) | `FORTRAN VAX` |
| | CDC | `FORTRAN CDC` |
| | Univac | `FORTRAN UNIVAC` |
| | IBM | `FORTRAN IBM` |
| | CRAY | `FORTRAN CRAY` |
| | CY205 | `FORTRAN CY205` |
| | *Machine* | `FORTRAN` *Machine* |
| Assembly | VAX (default) | `ASSEMBLY VAX` |
| | CDC | `ASSEMBLY CDC` |
| | Univac | `ASSEMBLY UNIVAC` |

*User keys* are additional distribution keys selected by the person that writes the deck for whatever purposes he or she has in mind. The only constraints placed on user keys are:

1. May contain any printable character except > < , # = / or *. Asterisks are used to identify macrokeys, *cf.* §5.1. Periods are reserved for cycled keys (§5.6). The first character must not be $ or ! because these are reserved for inline comments (§4.8).
2. May contain up to 24 characters.
3. Should not clash with implied key names.

User keys supplied in the command may contain *masking and cycle-range* specifications. This advanced use is explained in §5.8.

## §4.6 EMPTY DISTRIBUTION LIST: FILE SPLITTING

Can the distribution list be empty? Yes, but only in the stream mode when no user keys are specified. According to the rules of §4.3, nothing should be generated. But this case occurs so frequently in practice that the rule is reversed: *everything* is generated.

If everything is generated, what is the use of *MAX*? Simple answer: file splitting and selective deck extraction. Two examples:

```
$ MAX <EZGAL.VAX >/MSC
$ MAX <EZGAL.VAX >/FOR  #GMOPEN
```

The first *MAX* command splits the entire file EZGAL.VAX with MSC as common extension. The second extracts deck GMOPEN from EZGAL.VAX and puts it in file GMOPEN.FOR.

## §4.7 DECK HEADER MAPPING

In FOR or ASM mode, *MAX* "maps" the C=DECK line into a deck-header line appropriate to the target computer. To illustrate the mapping process, assume that the deck name is DEKNAM and that the deck type is FORTRAN. Then the header line format is as shown in the following table.

| *Target* | *Output header form* | *Comments* |
|---|---|---|
| VAX | C$FORTRAN DEKNAM | Innocuous editor "hook" |
| CDC | *DECK DEKNAM | For UPDATE utility |
| Univac | @FTN,SI DEKNAM | FTN element identifier |
| IBM | presently same as VAX | |
| CRAY | presently same as CDC | |
| CY205 | presently same as CDC | |
| *Machine* | presently same as VAX | |

These headers can be "massaged" with the text editor if necessary. For example, to expand all @FTN,SI to @FTN,SIO before submitting to Univac's ASCII FORTRAN compiler.

For ASSEMBLY and MIXED decks, the header line on Univac starts with @ASM,SI and @ELT,SI, respectively.

A similar mapping takes place for PROCEDURE-type decks that contain INCLUDE text. Look at the output and you'll get the idea.

## §4.8 *SYNTACTIC SUGAR

### Inline Comments

In an MSC statement line, anything after blank-dollar sign or blank-exclamation mark is treated as comment text.

### Other Control Statement Prefixes

In addition to `C=`, *MAX* recognizes the following control statement prefixes:

```
*=    %=    !=    .=
```

which are useful for exotic applications.

### The IF and ENDIF Variants of BLOCK and END

*MAX* treats the following control statements

```
C=IF        C=ENDIF
```

as equivalent to `C=BLOCK` and `C=END`, respectively. These statements provide enhanced readability in the use of the `ELSEIF` and `ELSE` control statements, which are discussed in §§5.2-5.4.

# 5

# MSC Files: Advanced Features

**IMPORTANT**: This section describes the more advanced features of *MAX*. Description relies heavily on specific examples. It is assumed that readers interested in these features can go from the particular to the general with only a modicum of help.

## §5.1 *MACROKEYS

Consider the following problem. A section of a machine-dependent FORTRAN subroutine contains code that works on both Univac and CDC, but not on other machines. How should the section be labeled with MSC statements? The brute-force solution is

```
C=BLOCK CDC

(code section)

C=END
C=BLOCK UNIVAC

(duplicated code section)

C=END
```

This is obviously unclean if the code section is substantial. The elegant solution is to use a macrokey:

```
C=*CU CDC .OR. Univac
C=BLOCK *CU

(code section)

C=END
```

The name of the macrokey in this example is CU, which stands for "either CDC or Univac". The name is picked by the code developer; it should not conflict with other key names, and must be defined before it appears in a C=BLOCK statement.

A macrokey definition statement may contain *only one type* of logical connective: AND or OR. Inasmuch as a macrokey definition may contain previously defined macrokeys, this is not much of a restriction, as the following example below shows.

**Caution**: There must be at least one blank space before and after an .OR. or .AND. connective.

Now suppose that a particular code section is valid only for two specific environments:

CDC computer, FTN4 FORTRAN compiler
Univac computer, FORTRAN V compiler

With the nested macrokey concept, this is easy to represent:

```
C=*C4 CDC .AND. FTN4
C=*U5 UNIVAC .AND. FORTRAN5
C=*C4U5 *C4 .OR. *U5
C=BLOCK *C4U5

(code section)

C=END
```

Here **FTN4** and **FORTRAN5** are names selected by the user; anything goes in this regard.

There are no limits on levels of macrokey nesting; however, in practice one level should be more than enough. (If you need more, your code is either too complex or machine dependent — simplify it.)

## §5.2 *THE ELSE CONTROL STATEMENT

Let's consider a variation on the previous theme. A section of a FORTRAN subroutine happens to work only on CDC. A "complementary" section works on everything *but* CDC. This can be elegantly expressed with the help of the C=ELSE statement:

```
C=BLOCK CDC

(CDC-restricted code)

C=ELSE

(code for other machines)

C=END
```

The code comprised between the ELSE and END will be written out if and only if CDC is *not* on the distribution list.

Here is another useful feature for FORTRAN subroutines that do floating-point arithmetic:

```
C=BLOCK DOUBLE
      double precision a(n), b(4,n)
C=ELSE
      real   a(n), b(4,n)
C=END DOUBLE
```

Unless DOUBLE is explicitly specified as a distribution key, the *single-precision* version is generated — if that's your intention.

## §5.3 *NEGATED KEYS

The C=ELSE effect discussed in §5.2 can be achieved in a more direct manner with *negated* keys. These are simply keys prefixed by a minus sign. For example:

```
C=BLOCK -UNIVAC

(code valid for any but Univac)

C=END -UNIVAC
```

meaning that the enclosed code block is written out unless key UNIVAC appears in the distribution list. This is the same as saying

```
C=BLOCK UNIVAC
C=ELSE

(code valid for any but Univac)

C=END UNIVAC
```

but saves one line of typing.

Negated keys may appear in macrokey definitions, as in

```
C=*SINGLE -DOUBLE .OR. CDC .OR. CRAY
```

with a "logical complement" meaning. It is also permissible to negate macrokeys, as in

```
C=BLOCK -*CU
```

User-specified distribution keys may also be negated. This feature has few practical uses for *MAX* execution, for why specify, say, -DEBUG, when the same effect is obtained by leaving out DEBUG in the first place? But it has more application when running *SCAM* (see §8.3).

## §5.4 *THE ELSEIF CONTROL STATEMENT

The most elaborate block-control structure accepted by *MAX* involves the combined use of BLOCK (or IF), ELSEIF, ELSE and END (or ENDIF) control statements. (Recall from §4.9 that C=IF is the same as C=BLOCK, and C=ENDIF the same as C=END.)

The structure strongly resembles the FORTRAN 77 IF-THEN-ELSE construct. This is again best explained by an example.

```
C=IF CDC

(code for CDC)

C=ELSEIF IBM

(code for IBM)

C=ELSEIF VAX

(code for VAX)

C=ELSE

("else" code)

C=ENDIF
```

Code for CDC will be written out if distribution key CDC appears. Code for IBM will be written out if distribution key IBM appears *and* CDC doesn't. Code for VAX will be written out if distribution key VAX appears *and* neither CDC nor IBM does. Finally, the "else" code will be written out if *none* of those three keys appears.

Note that if the keys appearing in this construction are ***mutually exclusive***, the order in which the subordinate code blocks appear is irrelevant; but it does matter if they are not. In practice this construction is best reserved for mutually exclusive keys or macrokeys. This is of course the case in the example, because the keys define different target computer environments.

A completely equivalent construction that uses only C=BLOCK and C=END is possible with negated keys (§5.3):

```
C=BLOCK CDC

     (code for CDC)

C=END CDC
C=BLOCK -CDC IBM

     (code for IBM)

C=END -CDC IBM
C=BLOCK -CDC -IBM VAX

     (code for VAX)

C=END -CDC -IBM VAX
C=BLOCK -CDC -IBM -VAX

     ("else" code)

C=END -CDC -IBM -VAX
```

This is not only less readable, but takes longer to type. The interesting point, however, is the way *MAX* internally handles ELSEIF and ELSE control statements: it maps them to C=BLOCK and C=END pairs according to the correspondence illustrated above, so it is in fact the latter form that is processed by *MAX*.

Some general comments. A C=ELSEIF must be followed by one and only one key or macrokey, while a C=ELSE takes none. The final C=END or C=ENDIF must be either key-less, or explicitly list all of the keys (or macrokeys) appearing in the matching C=BLOCK (or C=IF) and in the C=ELSEIFs.

Other C=BLOCKs may be nested arbitrarily deep within the code blocks shown above.

Macrokeys, C=ELSEIF, C=ELSE and nesting provide the adventurous user unlimited power on logically combining implied and user-defined keys to achieve any unusual effect.

## §5.5 *INCLUDE STATEMENT TRANSLATION

INCLUDE statements provide a structured and convenient way of specifying global symbols in FORTRAN programs, such as common blocks or PARAMETER declarations. Unfortunately FORTRAN 77 has not standardized the INCLUDE statement, and its format varies across compilers (if provided at all). Fortunately, the VAX-FORTRAN form

```
INCLUDE  'filename'
```

appears likely to become the standard in the next revision of the FORTRAN language† (perhaps without the apostrophe delimiters).

Because of the foregoing remark, *MAX* assumes that the VAX form of INCLUDE is "reference". This reference form may be exemplified by the VAX/FORTRAN statement

```
INCLUDE  'PROC:SYMBIO.PRC'
```

where the logical directory name and file extension (PROC and PRC in this example) are optional. When executing in FOR mode with the TI option on (§3.1), and the target machine is CDC or Univac, *MAX* translates the sample statement to

```
*CALL SYMBIO
INCLUDE SYMBIO
```

respectively. Translation is performed by stripping off the filename, and preceding it with INCLUDE for Univac or *CALL for CDC, with * in column 1. (On CDC, *CALL is an UPDATE directive, rather than a compiler recognizable statement.)

To make the translation process work smoothly, the following should be kept in mind. The word INCLUDE must appear in columns 7-13; all-uppercase or all-lowercase is acceptable. The filename *should not exceed six characters* for Univac compatibility. It is a good idea to keep all your INCLUDE text in one directory (PROC: in the example) because then it can be readily packed in one MSC file and translated as a block.

There is a separate converter utility (not *MAX*) which physically inserts INCLUDE text in the MSC file itself. This is useful if preparing FORTRAN code for shipping to a computer whose FORTRAN compiler does not provide an INCLUDE mechanism (for example, IBM). This utility is described in §7.

† Also called FORTRAN 8X.

## §5.6 *CYCLED KEYS

Three programmers, A. B. Cee, Jo. B. Queue and X. Y. Zee, are in charge of maintaining and updating a very large program, whose master source code is in file GODZILLA.MSC.

To keep track of who is doing what to which, the three agree on a "dated fingerprint" scheme to systematically mark updates and fixes before they are eventually incorporated in GODZILLA. They decide to use MSC blocks with "signature+date" keys. For example, a correction made by Mr. Cee on 7 November 1983 might be identified by the MSC statement

```
C=BLOCK ABC.31107 !  Fix relativistic viscoplasticity
    (code)
C=END ABC.31107
```

A user key of the form

*name.cycle*

is called a *cycled key*. Name is an ordinary string, but must not include periods. Cycle is an unsigned integer in the range 0 through 999999. Both components are separated by a period. (The statement after the exclamation mark is an inline comment, cf. §4.8.)

After some time of using this convention, GODZILLA.MSC may contain many cycled keys. Explicit specification of such keys in the distribution list may be inconvenient, and the *MAX* limit easily exceeded. This difficulty is eliminated by *masking* and *cycle-range* specifications in the user-specified distribution keys. The following examples should be sufficient to illustrate how these things work:

| *User key* | *Activates for distribution* |
|---|---|
| ABC* | All keys signed by ABC |
| ABC.3* | All 1983 keys signed by ABC |
| %%%.3* | All 1983 keys with 3-char signatures |
| *.3* | All 1983 keys |
| JBQ.306* | June 1983 keys signed by JBQ |
| JBQ.30100:30830 | Jan-Aug 1983 (incl) keys signed by JBQ |
| XYZ.:30830 | Pre-Sept 1983 keys signed by XYZ |
| %%%.30131: | All post-Jan 1983 keys |

Of course, the cycled-key scheme also fits less formal identification conventions, for example VERSION.1, VERSION.2, ... These may be attractive for one-person programs, for which a signature component is unnecessary.

# 6 REX

## §6.1 MOTIVATION

Consider the following situation. A large MSC file, SOLVER.MSC, contains over a hundred decks. Three of these: FACTOR, FORSUB and BACSUB, are split out into files FACTOR.DEK, FORSUB.DEK and BACSUB.DEK, respectively, and their code modified. After verifying the new software, the programmer wants to put the modified decks back into SOLVER.MSC. How to do this? On the VAX, you can follow three methods.

### Using the Text Editor

You can edit SOLVER.MSC in order to delete the text of the old decks and then invoke external-file copy to include the new decks. This approach is clumsy, time-consuming, and above all dangerous. It is easy to delete too much, or the wrong deck. Furthermore, external-file copy is very slow when editing large files.

### *MAX*split, Rename and Repack

This can be illustrated for the example case:

```
$ MAX     <SOLVER.MSC >/DDD
$ REN     FACTOR.DEK *.DDD
$ REN     FORSUB.DEK *.DDD
$ REN     BACSUB.DEK *.DDD
$ PUR
$ COPY    *.DDD  SOLVER.MSC
$ DEL     *.DDD.*
$ PUR
```

This approach is much safer than the text-editor method, and can be readily encapsulated in a command procedure. But splitting a big file can be quite slow, and the disk space of SOLVER.MSC triples during the process.

### Using *REX*

*REX* is a "deck replacement" utility which can be invoked very much like *MAX*. For the example case,

```
$ COPY FACTOR.DEK,FORSUB,BACSUB DECKS.DEK
$ REX <DECKS.DEK >SOLVER.MSC
```

*REX* produces *another cycle* of file SOLVER.MSC in which all decks that match those in DECKS.DEK — namely, FACTOR, FORSUB and BACSUB — are replaced by the latter. The replaced decks occupy the *same position* in the new MSC file as before.

If the three decks were in the same file to begin with, the first copy statement becomes unnecessary. If you plan to use *REX*, you may want to keep that fact in mind.

## §6.2 ACCESS AND USE INFORMATION

### Accessing *REX*

If the NICE system is officially installed in your VAX, you can gain access to *REX* simply by inserting the foreign-command definition

```
$ REX :==$NICE$EXE:REX.EXE
```

in your `LOGIN.COM` file. As always, the symbol on the left can be anything; here we shall assume `REX`.

If NICE is not installed on your VAX, the above command should reference the *REX* executable file located in one of your directories; see §2.1.

### Invoking *REX*

The format of the *REX*-invocation command is

```
$ REX [/Qualifiers] <Deckfile >MSCfile
```

where the file specifications are mandatory; if omitted, you will be prompted for them. Except for `REX` appearing first, command components can actually appear in any order.

The optional command qualifiers are:

| | |
|---|---|
| `APP` | Enable deck-append mode, as explained below. |
| `L` | Give run statistics. If omitted, *REX* stays silent except for error messages and a running commentary on decks replaced and/or appended. |

The file specifications are:

| | |
|---|---|
| *Deckfile* | Name of the file that contains the updated decks. Up to 24 decks are permitted; if more, the execution is terminated and no replacement is performed. If the file extension is omitted, `.MSC` is assumed. |
| *MSCfile* | The name of the MSC file that contains decks to be replaced. The output file has the same name but a higher version number (usually the next one). If the file extension is omitted, `.MSC` is assumed. |

**Caution**: Other than leaving out the extension, no file-name abbreviations are permitted.

## How Does *REX* Work?

During *REX* execution, each *MSCfile* deck whose name matches a deck name in *Deckfile* is replaced by the latter.

If a *Deckfile* deck does not match any deck name in the *MSCfile*, the outcome depends on whether the append mode has been specified with the APP qualifier. If the append mode is on, all unmatched *Deckfile* decks are written to the end of *MSCfile*. If the append mode is not on, nothing happens and an informative message to the effect that there were unmatched decks is given.

REMARK 6.1

If you always want the **APPEND** mode to be on when you invoke *REX*, you may redefine the foreign command as

```
$ REX :==$NICE$EXE:REX.EXE /APP
```

## Examples

```
$ REX/L         <UPDSTUFF.DDD >MARK6LIB.MSC
$ REX/L/APP     <BRAND.NEW  >CLIP.VAX
$ REX           <FACTOR.CMD >SOLVER.HLP
```

**Warning**: When it begins running, *REX* opens one scratch file for *each* deck present in the *Deckfile* before it begins scanning the *MSCfile*. For example, if the *Deckfile* has 20 decks, at a certain point there will be 23 files simultaneously open (20 for decks + *Deckfile* + input *MSCfile* + output *MSCfile*). If a *REX* execution terminates with an error message that says something about too many files being open at one time, have your computer system manager increase your "open file" quota and try again.

# 7 INCLUDE

## §7.1 THE *INCLUDE* UTILITY

### Purpose

*INCLUDE* is a utility processor that replaces INCLUDE statements as in VAX-FORTRAN (cf. §5.4) with actual text lines taken from the referenced files.

### Accessing and Executing *INCLUDE*

To make use of *INCLUDE* on a VAX with the NICE system installed, insert the foreign-command definition

```
$ IN :==$NICE$EXE:INCLUDE.EXE
```

in your LOGIN.COM file. (If the NICE system is not available on your VAX, proceed as indicated in §2.1.)

Once the command definition is activated, you can invoke *INCLUDE* as an input-output filter:

```
$ IN [/Qualifier] <Inputfile >Outputfile
```

The file specifications are mandatory; if omitted, the program will prompt you for them.

The only optional qualifier is L, which tells INCLUDE to print run statistics on completion.

The input and output filenames are specified in the usual way. The output filename may be abbreviated to the version portion only, for example >.FOR. For verification purposes terminal output may be specified by >O.

**Important.** *INCLUDE* is not restricted to processing MSC files; in fact, it pays no attention whatsoever to MSC statements.

### Example

```
$ IN/L <CLIP.VAX >CLIP.IBM
```

# 8 SCAM

## §8.1 THE *SCAM* UTILITY

### Motivation

A program developer that maintains a large program file, say ADVLAM.MSC, would like to do the following:

> "Find and list on the screen all source lines included in each MSC control block that has the key UPDATE.4"

Trying to do this with *MAX*, for example by saying

```
$ MAX <ADVLAM >O UPDATE.4
```

will *not* usually work. The trouble is that the distribution process depends on the way in which key UPDATE.4 "telescopes" with regards to other keys. But for the foregoing request key interrelations are irrelevant. Only the UPDATE.4 key, by itself, matters.

This key-lookup job can be conveniently done with another *MAX* helper called *SCAM* (a contraction of "SCAn Msc"). First, as usual, a foreign command definition is inserted in your LOGIN.COM file:

```
$ SCAM :==$NICE$EXE:SCAM.EXE
```

assuming again that NICE is available to you; if this is not the case please read again §2.1. Once the definition is activated, the command

```
$ SCAM <ADVLAM  >O UPDATE.4
```

will work. The output from *SCAM* will display the requested blocks appropriately identified (in cols. 73-80) by the deck name and deck line number. It should look something like this:

```
C=BLOCK UPDATE.4                                  MASTER
      x = colblk (mat, n, m, 1)                   000343
      y = sqrt(x)                                 000344
C=END UPDATE.4                                    MASTER
```

In this example the *SCAM* output shows lines 342 through 345 of deck MASTER.

## §8.2 HOW TO *SCAM*

The general form of the *SCAM*-invocation command is

```
$ SCAM [/Qualifiers] <Inputfile >Outputfile Scankeys [#Decknames]
```

where

| | |
|---|---|
| *Qualifiers* | The optional qualifiers are: L, which causes run statistics to be printed, and NOID (which may be abbreviated to N) to suppress deck identification text in columns 73-80. |
| *Inputfile* | Name of the MSC file to be scanned. If omitted, you will be prompted for it. If the file extension and prefix period are omitted, .MSC is assumed. |
| *Outputfile* | Name of the file that will receive *SCAM* output. If omitted, you will be prompted for it. Often set to zero to get output on the screen; an actual name may be used, however, to create a file that is then sent to a printer or inserted in a document. |
| *Scankeys* | A list of MSC-block keys to be scanned for. Keys do not interact; the output will be simply the *union* of the outputs for each scan key. At least one key must be specified; if none given, you will be prompted for *one* key. |
| *Decknames* | Optionally, a list of deck names to which the scanning process will be restricted. The specification format is the same as for *MAX*. |

### Examples

```
$ SCAM <SOLVER >.SCN CDC
$ SCAM/L <SOLVER >O VAX UNIVAC
$ SCAM/N <CLIP.VAX >O MACRO #CLM*
$ SCAM/L <CLIP.VAX >O DECK
```

REMARK 8.1

SCAM recognizes no implied keys, as MAX does. To display all FORTRAN lines, for example, the scan key FORTRAN must be given explicitly in the SCAM invocation.

REMARK 8.2

The output will also contain macrokey-controlled C=BLOCKs if a scan key intervenes in the macrokey definition.

REMARK 8.3

If C=BLOCKs are followed by C=ELSEIFs or C=ELSEs, negated scan keys may be used to discern the internal structure (see §8.3).

REMARK 8.4

One-line control statements such as `C=PURPOSE`, `C=AUTHOR`, etc., which are ignored by *MAX*, are *not* ignored by *SCAM*. For example, the following command will display on the screen all `C=PURPOSE` lines (and nothing more) included in MSC file `ADVLAM.MSC`:

```
$ SCAM <ADVLAM >O  PURPOSE
```

## §8.3 *ADVANCED SCAMMING

This subsection covers some of the more advanced features of *SCAM*.

### Listing C=DECK Lines

Special scan key **DECK** causes **C=DECK** lines to be listed. No identification field appears. If you do this sort of thing often, it is convenient to define a one-line procedure

```
$ SCAM <'P1' DECK >O
```

whose execution is abbreviated to, say, **LD**. Then

```
$ LD  SOURCE.MTR
```

lists all deck lines (and nothing else) in **SOURCE.MTR**.

### Macrokeys

Special scan key ** causes all macrokey-definition lines to be displayed.

### Negated Keys

To list all CDC-dependent code lines in **CLIP.VAX** you would say

```
$ SCAM <CLIP.VAX >O  CDC
```

That was easy. Now, how about all "other than CDC" code lines in blocks that mention CDC? Use the *negated* scan key:

```
$ SCAM <CLIP.VAX >O  -CDC
```

So there is some use for negated distribution keys, after all.

# 9
# A Trip Down Memory Lane

## §9.1 MAXIFYING OLD CODE

By now you are hopefully sold on the virtues of *MAX and Friends*. You have probably noticed that using Master Source Code on *new* software developments is straightforward. The first operation in making a new deck usually consists of copying an existing one; recurring descriptive constructs such as `PURPOSE`, `ABSTRACT` and `USAGE` are edited, and MSC control statements inserted while the source code is written. But how about old FORTRAN programs brought from other computers or your old tapes?

If these programs are fairly large, the conversion to MSC form can be laborious. Fortunately there are a couple of utilities that can reduce the work by getting you started with the minimum necessary to use *MAX*. These two utilities, *PUTDECK* and *EZ2MAX*, are described in the following subsections.

## §9.2 INSERTING DECK IDENTIFIERS

To use *MAX*, decks have to be labelled with C=DECK lines. This is the irreducible minimum: once deck identifiers are in place you can at least split files and extract decks. This insertion can be automatically done with the *PUTDECK* utility.

The first step is to merge all program units (subroutines, functions) into a ***single file.*** (This step is only needed if the code comes from a machine such as Univac, which keeps programs fragmented into elements.) A masked copy such as

```
$ COPY *.FOR BIGFILE.MER
```

should make this happen.

Next, insert the foreign-command definition

```
$ PD:==$NICE$EXE:PUTDECK.EXE
```

in your LOGIN.COM, and activate it with a @LOGIN. (If NICE is not in your Vax, read §2.1.) Finally, execute *PUTDECK* as an input-output filter, for example

```
$ PD  <BIGFILE.MER  >.MSC
```

This execution inserts C=DECK lines *immediately before* each subroutine or function statement. The name of the subroutine or function becomes the deck name. Decks are typed as FORTRAN. (If you don't like these rules, you can always edit the output file.)

### *PUTDECK* Restrictions

To ensure successful operation of *PUTDECK*, you must keep certain restrictions in mind.

The input file must be without any MSC control statements. Never use *PUTDECK* on a "partly-MSCed" file.

The input file should be free of parasite control cards brought from other machines, *e.g.*, Univac's @FORs and @ELTs.

In subroutine-declaration lines, the word SUBROUTINE must be followed by at least one blank; furthermore, columns 1-6 of that line must be blank. The word SUBROUTINE need not start in column 7, however, and may be in lower or upper case. The subroutine name need not be terminated by a blank, but must be in the same line as the SUBROUTINE declaration. The argument list need not start in the same line. Thus the following declarations are acceptable:

```
SUBROUTINE SOLVE(A,B,C)

Subroutine   Solve   (A, B, C)
```

The following, however, is not acceptable

```
SUBROUTINESOLVE (A,B,C)
```

and will be sadly missed by *PUTDECK*.

Very similar constraints apply to function-declaration code lines. The word **FUNCTION** must be followed by a blank, and the function name must be in the same line. The word **FUNCTION** may be preceded *in the same line* by one of the following type qualifiers

```
CHARACTER
DOUBLE PRECISION
COMPLEX
INTEGER
LOGICAL
REAL
```

If any of these qualifiers appear, they must be delimited by one or more blanks. For example the function declaration

```
DOUBLEPRECISION FUNCTION
```

will be missed.

The **CHARACTER** function qualifier may be followed by a length specification such as CHARACTER*4 or CHARACTER*(*). Lower or upper case is acceptable.

Two more limitations should be mentioned. *PUTDECK* does not recognize main programs, **BLOCK DATA** units, and **INCLUDE** text. These have to be done by hand.

If in doubt as to whether the input file complies with these requirements, look at the *PUTDECK* output with the text editor or with *SCAM*, correct the source file as needed, and try again.

## §9.3 INSERTING FORTRAN BLOCKS

If you are satisfied with just the C=DECK lines, you need not proceed further. Now you can at least split out decks with *MAX* and put them back in with *REX*.

To go a step further, insert and activate the definition

```
$ E2M:==$NICE$EXE:EZ2MAX
```

then execute *EZ2MAX* as an input-output filter, as in

```
$ E2M  <BIGFILE.MSC  >FORFILE.MSC
```

This inserts C=BLOCK FORTRAN and C=END FORTRAN lines after each C=DECK line and before the next C=DECK line, respectively. Now you can try MAX/F.

# A

# Assembly Code Maintenance

## §A.1 TARGET AUDIENCE

This Appendix has been prepared for the minority of "real programmers" that maintain sizable amounts of assembly language code for CDC, IBM, Univac, Cray or VAX computers. All-FORTRAN programmers need not be aware of this material.

With the advent of FORTRAN 77 the need for assembly language programming has drastically diminished. There are three areas, however, in which assembly language can be useful:

1. To do things not provided for in the FORTRAN 77 language and unavailable as FORTRAN extensions. Example: get the machine address of a variable in IBM FORTRAN.

2. To speed up time-critical sections of number-crunching programs when assembly code make possible to fully exploit hardware features such as pipelining.

3. To reference system level input/output facilities otherwise inaccessible. Example: the RMS (Record Management Services) level of VAX/VMS.

If you keep a few assembly language routines around that are hardly ever changed, there is little motivation for putting such routines into MSC files. But if you keep a substantial amount of assembly code in support of specific application packages, the source should be kept just like FORTRAN, *i.e.*, neatly packaged in MSC files on the VAX. This approach centralized maintenance and lets you rapidly extract versions for shipping to other machines.

Packaging assembly code in MSC files requires some care, however, as some surprises may otherwise result. The purpose of this Appendix is to explain how to proceed so you get it right the first time.

## §A.2 ALL-ASSEMBLY MSC FILES

The simplest MSC organization of assembly code is one in which

(1) All decks contain *only* assembly code; these are called ASSEMBLY decks.

(2) MSC files contain only ASSEMBLY decks.

This organization is recommended if you keep just a moderate number of general purpose assembly language routines around. Putting them together in one MSC file simplifies keeping track of where they are.

Later subsections in this Appendix discuss two more general organizations: keeping ASSEMBLY and FORTRAN decks in the same MSC file (§A.3), and keeping FORTRAN and assembly code in the same deck (§A.4).

### Structure of ASSEMBLY Decks

The basic structure of an ASSEMBLY deck identified as *Deckname* for a target computer identified by key *Machine* is

```
C=DECK Deckname Deckname ASSEMBLY
C=BLOCK Machine ASSEMBLY

     (assembly code)

C=END Machine ASSEMBLY
```

To illustrate this structure, here is an "unadorned" assembly language routine for Univac, written by somebody that obviously does not believe in comments:

```
C=DECK RW RW ASSEMBLY
C=BLOCK UNIVAC ASSEMBLY
      AXR$                       .
$(1)                             .
      LA      AO,*0,X11          .
      SA      AO,PKT             .
      LA      AO,*1,X11          .
      SA      AO,PKT+1           .
      SZ      PKT+2              .
      SZ      PKT+3              .
      LA      AO,2,X11           .
      SA,1    AO,PKT+4           .
      LA      AO,*3,X11          .
      SA,2    AO,PKT+4           .
      LMA     AO,*4,X11          .
      SA      AO,PKT+5           .
      LA      AO,*5,X11          .
      SA,12   AO,PKT+3           .
      LA,U    AO,PKT             .
      ER      IOW$               .
      LA,13   AO,PKT+3           .
      SA      AO,*6,X11          .
      LA,1    AO,PKT+3           .
      SA      AO,*7,X11          .
      J       9,X11              .
$(0)                             .
PKT   RES     10                 .
      END
C=END UNIVAC ASSEMBLY
```

This example shows the minimum number of MSC control lines for an ASSEMBLY deck, namely three.

REMARK A.1

The above code is compatible with Univac's ATHENA FORTRAN (also called FORTRAN V) compiler. A different version of RW is required for Univac's ASCII FORTRAN compiler, which implements the FORTRAN 77 standard. An example that illustrates packaging for both compilers appears later in this subsection.

The following ASSEMBLY deck does have the recommended level of documentation:

```
*=DECK LOCF LOCF ASSEMBLY
*=BLOCK IBM ASSEMBLY
*=PURPOSE Get absolute machine address of argument (IBM)
*=AUTHOR F. A. Weiler
*=VERSION July 1982
*=KEYWORDS get absolute argument address
*=EQUIPMENT IBM
*=BLOCK ABSTRACT
*
*    LOCF is a function that returns the absolute address of its
*    argument.  On CDC, Univac and Vax, this is an inline
*    function provided by the FORTRAN compiler, and is named
*    LOCF, LOC and %LOC, respectively (LOCF and LOC return a
*    WORD address, whereas %LOC returns a BYTE address).
*    IBM FORTRAN does not provide such a function, hence this
*    assembly routine generously contributed by Frank Weiler.
*
*=END ABSTRACT
*=BLOCK USAGE
*    The function reference is:
*
*         IADDR = LOCF (ARG)
*
*    and IADDR receives the absolute address of ARG in BYTES.
*
*=END USAGE
LOCA   CSECT
RO     EQU     0            WORKING REGISTER
R1     EQU     1            WORKING REGISTER
R13    EQU     13           COMMUNICATION REGISTER
R14    EQU     14           COMMUNICATION REGISTER
R15    EQU     15           PROGRAM ADDRESSING REGISTER
       ENTRY   LOCF
LOCF   DS      OA           DEFINE START OF SUBROUTINE
       L       RO,O(R1)     ADDRESS OF ARGUMENT TO RO
       BR      R14          RETURN TO CALLING ROUTINE
       END
*=END IBM ASSEMBLY
```

REMARK A.2

The use of *= in lieu of C= has no deep significance; *= is always equivalent to C= and * happens to be the column-1 comment character for IBM assembly code.

REMARK A.3

It is always a good idea to make the **C=BLOCK** *Machine* **ASSEMBLY** the *second* line of the deck. This makes it easier for *MAX* to avoid writing empty decks to the output file when **ASSEMBLY** and **FORTRAN** decks are intermixed (§A.3).

Can the same ASSEMBLY deck hold assembly code for more than one machine? Yes. Just stack C=BLOCKs as illustrated next:

```
C=DECK FAKER FAKER ASSEMBLY
C=BLOCK UNIVAC ASSEMBLY

     (Univac assembly code)

C=END UNIVAC ASSEMBLY
C=BLOCK IBM ASSEMBLY

     (IBM assembly code)

C=END IBM ASSEMBLY
```

Another way of doing it, with the IF ... ELSEIF ... ENDIF construct:

```
C=DECK FAKER FAKER ASSEMBLY
C=BLOCK ASSEMBLY

C=IF UNIVAC

     (Univac assembly code)

C=END UNIVAC
C=ELSEIF IBM

     (IBM assembly code)

C=ENDIF
C=END ASSEMBLY
```

Here the use of the IF ... ENDIF does not necessarily make things more readable if assembly sections are fairly long. Personally I prefer the first form.

## Multicompiler Packaging

Not only is assembly code machine dependent but sometimes within the same computer the subroutine linkage changes as another FORTRAN compiler appears on the scene. Such change has the unfortunate consequence of requiring multiple versions of the same assembly routine to be maintained (at least until the old compiler finally disappears). For Univac, the appropriate packaging for say, RW would be as follows:

```
C=DECK RW RW ASSEMBLY
C=BLOCK UNIVAC ASSEMBLY
C=IF ATHENA

    (ATHENA-FORTRAN-compatible version)

C=ELSE

    (ASCII-FORTRAN-compatible version)

C=ENDIF
C=END UNIVAC ASSEMBLY
```

Use of the identifier ATHENA is not mandatory but is recommended because *MAX* looks for it when generating deck-header control cards for Univac.

For other computers the recipe is the same. Fortunately, it is not presently needed for CDC (the old RUN compiler is no longer used), VAX or Cray; as for IBM I don't know.

## Univac Code Extraction

To extract the Univac version from an all-assembly MSC file, the following *MAX* commands are recommended:

| *Compatible with* | |
|---|---|
| ASCII FORTRAN | $MAX/A/UNI/WC/HDG <*Inputfile* >*Outputfile* |
| ATHENA FORTRAN | $MAX/A/UNI/WC/HDG <*Inputfile* >*Outputfile* ATHENA |

REMARK A.4

The ATHENA distribution key is assumed (see **Multicompiler Packaging**); if you use a different key replace it in the second invocation form.

REMARK A.5

Each nonempty output deck is preceded by @ASM,SI so that the output file can be directly @ADDed to the runstream once it has been shipped to Univac.

REMARK A.6

The WC qualifier forces comment-only lines to be written out. Most assembly code is (or should be) commented line by line, so sending comment-only lines probably does not make much difference.

REMARK A.7

A `@HDG,P` line bearing the deck name is generated before each `@ASM,SI`; if you don't want this remove the `HDG` qualifier.

## CDC Code Extraction

To extract the CDC version from an all-assembly MSC file you should use a *MAX*-invocation command such as

```
$ MAX/A/CDC/WC <Inputfile >Outputfile
```

The output file is a multideck file suitable for input to CDC's UPDATE utility. The COMPILE output from UPDATE can be presented to either the COMPASS assembler or to the FORTRAN compiler (more about this choice in §A.3).

## VAX Code Extraction

To extract the VAX version from an all-assembly MSC file you should use

```
$ MAX/A/WC <Inputfile >/Extension
```

For example:

```
$ MAX/A/WC <UTILITY.ASM >/MAR
```

Note the *split output* specification. Each VAX assembly deck thereby becomes a separate file. These files can then be processed by VAX/VMS's MACRO assembler through a system command such as

```
$ MAC COMPARE,PRODUCT,SUM
```

which assume a `.MAR` extension. This approach is necessary because MACRO, unlike the FORTRAN compiler, cannot assemble multiple modules contained in one file; it quits on detecting an `.END` line.

REMARK A.8

To circumvent the multiple-file problem you could keep all VAX assembly code stacked in one deck, with a single `.END` card. this is not recommended, however, if the total code is voluminous as maintenance would be hindered.

## §A.3 MIXING FORTRAN AND ASSEMBLY DECKS

Most assembly code is written in support of specific application or utility packages; for example a terminal reader in support of a command language interpreter. If the FORTRAN package is merged into a MSC file, it is natural to include the supporting assembly code with it.

Intermixing ASSEMBLY and FORTRAN decks should cause no problems. The only thing to watch for is uniqueness in deck names. In this regard you are reminded that the deck name does not necessarily have to be the same as the entry point name.

As an example of this organization, suppose that VAX file MELANGE.MSC contains

```
C=DECK APPEND
C=BLOCK FORTRAN

    (FORTRAN code)

C=END FORTRAN
C=DECK FASTPROD FASTPROD ASSEMBLY
C=BLOCK CDC ASSEMBLY

    (CDC assembly code)

C=END CDC ASSEMBLY
C=BLOCK UNIVAC ASSEMBLY

    (UNIVAC assembly code)

C=END UNIVAC ASSEMBLY
C=BLOCK VAX ASSEMBLY

    (VAX assembly code)

C=END VAX ASSEMBLY
    . . .
    . . .

C=DECK ZZZ
C=BLOCK FORTRAN

    (FORTRAN code)

C=END FORTRAN
```

In the *stream mode*, a file like this is processed just like any other MSC file, because in the stream mode *MAX* ignores deck types. The difference comes when FORTRAN or Assembly mode is used. The invocation

```
$ MAX/F ...
```

extracts only FORTRAN decks and ignores ASSEMBLY decks, so it works as if ASSEMBLY decks were not physically there. For MELANGE.MSC, deck FASTPROD would be "transparent". On the other hand,

```
$ MAX/A ...
```

extracts only ASSEMBLY decks and ignores FORTRAN decks, so it works as if FORTRAN decks were not physically there. For MELANGE>MSC, decks APPEND and ZZZ would be "transparent".

The *combined* specification

```
$ MAX/A/F ...
```

processes both FORTRAN and ASSEMBLY decks in one pass. No deck is transparent. This makes sense for two target machines: Univac and CDC, but for different reasons.

## MAX/A/F for Univac

The output file receives extracts of both FORTRAN and ASSEMBLY decks, in the same order as they appear in the MSC file, preceded by appropriate control cards. For the example file the command

```
$ MAX/A/F/UNI/WC/HDG <MELANGE >.UNV
```

will give you MELANGE.UNV, which looks like

```
@HDG,P                  A P P E N D
@FTN,SIO APPEND

     (ASCII FORTRAN code)

@HDG,P                  F A S T P R O D
@ASM,SI FASTPROD

     (ASCII FORTRAN-compatible assembly code)

                        ...
                        ...
@HDG,P                  Z Z Z
@FTN,SIO ZZZ

     (ASCII FORTRAN code)
```

Although in Univac the FORTRAN compiler and the Assembler are separate system processors, a file configuration such as MELANGE.UNV makes sense because it embeds the control cards which specify which processor is to be called. Consequently, MELANGE.UNV can be @ADDed to the runstream as one file.

REMARK A.9

If the **ATHENA** distribution key is specified, all **@FTN,SIO** become **@FOR,SI**.

### MAX/A/F for CDC Cyber

If the example file is processed by

```
$ MAX/A/F/CDC/WC <MELANGE >.CDC
```

you get MELANGE.CDC, which looks like

```
*DECK   APPEND

        (CDC FORTRAN code)

*DECK   FASTPROD

        (COMPASS code)

                             ...
                             ...
*DECK   ZZZ

        (CDC FORTRAN code)
```

Note that the resulting output contains intermixed FORTRAN and Assembly code but no control cards; only UPDATE cards. Merging does make sense, however, because of a unique feature of CDC FORTRAN: both the FTN (FORTRAN 66) and FTN5 (FORTRAN 77) compilers can detect and process interspersed COMPASS assembly code as long as such code is properly identified by an IDENT line with the word IDENT starting at column 11.

Another interesting consequence of this CDC capability is the fact that COMPASS code that implements features unique to CDC can be maintained in FORTRAN decks if you wish. For example:

```
*=DECK FL
*=BLOCK CDC FORTRAN
       IDENT    FL (K)
*
* SUBROUTINE FL(K)
* RETURN FIELD LENGTH IF K = 0
* SET FIELD LENGTH TO K IF K = 0
*
       ENTRY    FL
FL     BSS      1
       SA2      X1                 X2=K
       BX7      X1                 X7=ADS(K)
       LX2      30                 SHIFT K TO LEFT HALF
       SA7      TEMP               SAVE ADS(K) IN TEMP
       BX6      X2                 X6=SHIFTED K
       SA6      STAT               STORE IN STAT
       MEMORY   CM,STAT,RECALL     MACRO FOR FIELD LENGTH
       SA1      STAT               X1=STAT
       SA2      TEMP               X2=TEMP
       AX1      30                 MOVE STAT TO RIGHT HALF
       BX6      X1                 RESULT TO X6
       SA6      X2                 RETURN STAT TO K
       EQ       FL                 RETURN
TEMP   BSS      1
STAT   BSS      1
       END
*=END CDC FORTRAN
```

FL is a good candidate for being in a FORTRAN deck because it implements a programming function unique to CDC equipment. The fact that a * in column 1 also identifies comment lines in FORTRAN 77 is quite useful because *MAX*'s WC qualifier acquires uniform implementation.

## §A.4 MIXED DECKS

A deck that contains both assembly code and FORTRAN code is called a MIXED deck, and must be identified as such in the C=DECK line. This deck organization may be used (with caution!) when you have the same subroutine or function implemented on both FORTRAN and assembly code for some machines.

To give an example, suppose that you have CDC and Univac assembly versions of subroutine ABORT (which forces abnormal run termination) and also a more-or-less machine-independent version implemented in FORTRAN. This can be collectively packaged into one deck with the following structure:

```
C=DECK ABORT ABORT MIXED
C=IF CDC
C=BLOCK ASSEMBLY

        (CDC assembly code)

C=END ASSEMBLY
C=ELSEIF UNIVAC
C=BLOCK ASSEMBLY

        (Univac assembly code)

C=END ASSEMBLY
C=ELSE
C=BLOCK FORTRAN

        (FORTRAN version)

C=END FORTRAN
C=ENDIF
```

The IF ... ELSEIF ... ELSE ... ENDIF organization is a foolproof one for MIXED decks in spite of the fact that it is somewhat difficult to read. Here are the basic rules:

1. Assembly versions go within ELSEIF blocks except the first one, which goes in the IF block.
2. The C=IF statement must be the second line of the deck.
3. Ordering of the assembly versions is irrelevant, but multicompiler versions, if any, must be properly nested (this is not shown in the example).
4. The FORTRAN version is always the last one, and it goes in the ELSE block.

## An Alternative Structure

There is a seemingly simpler deck organization that does not use the IF ... ENDIF construct:

```
C=DECK ABORT ABORT MIXED
C=BLOCK CDC ASSEMBLY

    (CDC assembly code)

C=END CDC ASSEMBLY
C=BLOCK UNIVAC ASSEMBLY

    (Univac assembly code)

C=END UNIVAC ASSEMBLY
C=BLOCK FORTRAN

    (FORTRAN version)

C=END FORTRAN
```

But this has a flaw. Can you think what it is? Answer: if you try to MAX/A/F this deck for Univac or CDC, both the assembly *and* the FORTRAN version will be written to the output file, which will get you some nasty Assembler diagnostics. This problem can be circumvented by qualifying the FORTRAN block with negative keys:

```
C=DECK ABORT ABORT MIXED
C=BLOCK CDC ASSEMBLY

    (CDC assembly code)

C=END CDC ASSEMBLY
C=BLOCK UNIVAC ASSEMBLY

    (Univac assembly code)

C=END UNIVAC ASSEMBLY
C=BLOCK FORTRAN -CDC -UNIVAC

    (FORTRAN version)

C=END FORTRAN -CDC -UNIVAC
```

but this is not very readable. And if you later on insert another assembly block, say for VAX, you have to remember to add -VAX to the FORTRAN block.

NASA
National Aeronautics and Space Administration

Report Documentation Page

| | | |
|---|---|---|
| 1. Report No. NASA CR-178383 | 2. Government Accession No. | 3. Recipient's Catalog No. |
| 4. Title and Subtitle Utilities for Master Source Code Distribution: MAX and Friends | | 5. Report Date October 1988 |
| | | 6. Performing Organization Code |
| 7. Author(s) Carlos A. Felippa | | 8. Performing Organization Report No. LMSC-D812789 |
| 9. Performing Organization Name and Address Lockheed Missiles and Space Company, Inc. Research and Development Division 3251 Hanover Street Palo Alto, California 94304 | | 10. Work Unit No. 505-63-01-10 |
| | | 11. Contract or Grant No. NAS1-18444 |
| 12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225 | | 13. Type of Report and Period Covered Contractor Report |
| | | 14. Sponsoring Agency Code |

15. Supplementary Notes
Current affiliation: Carlos A. Felippa, Center for Space Structures and Controls, Campus Box 429, University of Colorado, Boulder, CO 80309-0429

Langley Technical Monitor: W. Jefferson Stroud

16. Abstract
MAX is a program for the manipulation of Fortran master source code (MSC). This is a technique by which one maintains one and only one master copy of a Fortran program under a program development operating system, which for MAX is assumed to be VAX/VMS. The master copy is not intended to be directly compiled. Instead it must be pre-processed by MAX to produce compilable instances. These instances may correspond to different code versions (for example, double precision versus single precision), different machines ( for example, IBM, CDC, Cray) or different operating systems (for example VAX/VMS versus VAX/UNIX). The advantages of using a master source is more pronounced in complex application programs that are developed and maintained over many years and are to be transported and executed on several computer environments. The "version lag" problem that plagues many such programs is avoided by this approach. MAX is complemented by several auxiliary programs that perform nonessential functions. The ensemble is collectively known as MAX and Friends. All of these programs, including MAX, are executed as foreign VAX/VMS commands and can be easily hidden in customized VMS command procedures.

| | |
|---|---|
| 17. Key Words (Suggested by Authors(s)) Software engineering utilities Source code maintenance | 18. Distribution Statement Unclassified—Unlimited Subject Category 39 |

| | | | |
|---|---|---|---|
| 19. Security Classif.(of this report) Unclassified | 20. Security Classif.(of this page) Unclassified | 21. No. of Pages 68 | 22. Price A04 |

NASA FORM 1626 OCT 86
For sale by the National Technical Information Service, Springfield, Virginia 22161-2171