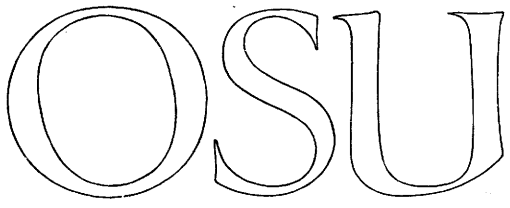
cc-69-1

DEFINE and DIRECT
(for CDC 3300/OS-3)

by
Gilbert A. Bachelor



COMPUTER CENTER
Oregon State University
Corvallis, Oregon 97331

DEFINE and DIRECT

(for CDC 3300/OS-3)

cc-69-1

by

Gilbert A. Bachelor

Computer Center
Oregon State University
Corvallis, Oregon 97330

DEFINE and DIRECT
(for CDC 3300/OS-3)

DEFINE and DIRECT are two programs that are stored in public files (under the names *DEFINE and *DIRECT), which are available to all users of OS-3. DEFINE is called by a control statement of the form [*DEFINE,arguments]. It can be used to equip and unequip units and files, to protect and unprotect files, to save and delete files, and to change the names of files. At the option of the user, DEFINE and DIRECT can be used to maintain a (private) file containing the names of other saved files belonging to the user. We shall first describe DEFINE, then DIRECT and the directory feature.

The *DEFINE statement.

The things which are done by a *DEFINE statement can be done by one or more of the OS-3 control statements EQUIP, UNEQUIP, FP, RFP, SAVE, and DELETE. These statements allow only one argument, but *DEFINE allows many arguments in one statement. Thus, one *DEFINE statement can do the work of many other statements. For example:

```
*DEFINE,10,5=LP,75=DATA3,56=BINDECK$,25+,40/,2-
```

This statement illustrates some of the things that a*DEFINE statement can do. In order, it equips logical unit 10 as a file, equips 5 as a line printer, equips 75 to a saved file DATA3, saves unit 56 under the name BINDECK, protects unit 25, unequips unit 40, and removes protection from unit 2. It would take at least seven OS-3 control statements to do all of this, and possibly more. If units 10, 5, or 75 were already equipped, one would first have to UNEQUIP them; DEFINE does this automatically. Also, in 75=DATA3, if there is no saved file with the name DATA3, *DEFINE will equip 75 as a file and save it under the name DATA3, And, if unit 40 were a non-saved, protected file, one would first to remove protection (RFP) before unequipping it; again, *DEFINE does this for the user.

Here is another example:

*DEFINE,12=95,ZAP/,15=QUORK?,1=PVZ-,PRG/*PROG

This *DEFINE statement equips unit 12 to unit 95 (unequipping 12 first, if necessary), deletes a saved file name ZAP, equips 15 to a saved file QUORK only if it exists, equips 1 to saved file PVZ and removes protection from it, and changes the name of saved file PRG to *PROG.

These two examples illustrate the notations used in the *DEFINE statement. In general, arguments are of five basic forms, they may be followed by one of 5 "suffix" control characters, and they are separated by commas. The suffix control characters may modify the meaning of the basic argument, or they may simply specify an additional action to be performed. The suffixes and their general meanings are:

+ Protect a file.

- Remove protection.

$ Save file under specified name.

? Equip file only if it exists; or, an inquiry: does the file exist?

/ Unequip, or delete.

The five basic forms of arguments are listed in the table below, which also shows which suffixes are allowed with each form of argument. An "X" means that the suffix is allowed. (lun) and (lun1) represent logical unit numbers in the range 0 to 99. (lun2) is a logical unit number in the range 0 to 100. (name), (name1), and (name2) represent names consisting of one or more letters, digits, and/or asterisks, in which the first character is a letter or an asterisk. These names may be hardware types (LP, PUN, etc.), or they may be names of saved files.

Form of argument	Suffixes allowed +	-	$	?	/
lun	X	X			X
name	X	X		X	X
lun1=lun2	X	X			
lun=name	X	X	X	X	
name1/name2	X	X			

Each form of argument may be used without a suffix, or with one of the suffixes indicated above. In all, there are 20 combinations allowed. We shall list these below, and describe the meaning of each. First, we give a list of error messages which DEFINE may print out. The numbers in the descriptions of the arguments refer to the numbers of the error messages in the following table.

Error messages.

1. *SYSTEM ERROR* This message means that an error has occurred in OS-3 or in DEFINE; it may be due to either a hardware or a software malfunction.
2. ILLEGAL ARGUMENT An argument has an improper form. The argument is ignored, and DEFINE goes on to process the next one (if any).
3. (lun) IS NOT DEFINED The specified (lun) is not equipped.
4. (name) IS BUSY The file (name) is being used by someone else.
5. NOT ENOUGH FILE SPACE FOR (name) Either there is not enough saved file space available to save the file; or, there is not enough scratch file space available to delete the file.
6. (name) IS NOT A FILE (name) is not a saved file. This message occurs if one tries to protect, or to delete, a "file", using a "hardware" name.
7. (lun) IS NOT A FILE This message occurs if one tries to save, or to protect, a unit that is not a file.
8. (name) IS PUBLIC This message occurs if one tries to delete, or to remove protection from, a public file that belongs to someone else.
9. NAME NEEDED WITH (lun) The specified (lun) is a saved file; one must specify its name in order to remove protection.
10. (name) DOES NOT EXIST There is no saved file with the specified name.
11. (lun) IS ALREADY SAVED The (lun) is already a saved file and cannot be saved under another name.

12. (name) ALREADY EXISTS There is already a saved file with the specified name; one cannot save another file under the same name.

13. ILLEGAL NAME: (name) The specified (name) is a hardware type; it is illegal to save a file under such a name.

Note: The following names are hardware types:

FILE, LP, MT, NULL, PLOT, PR, PUN, RAF.

There are no hardware names for "card reader", "teletype", or "display console". These are "standard input devices", and whichever one of these is available to the user is always equipped as logical unit 100.

Arguments and their meanings.

(lun) Unequip (lun) and equip it as an empty scratch file. Errors: none.

(lun)+ Protect (lun). Errors: 3, 7.

(lun)- Remove protection from (lun). Errors: 3, 9.

(lun)/ Unequip (lun), even if it is a protected, non-saved file. Errors: none.

(name) If there is no saved file with the specified name, create an empty saved file with this name. Errors: 4, 5.

(name)+ Protect the saved file with the specified name. Errors: 4, 6, 10.

(name)- Remove protection from the specified saved file. Errors: 4, 8, 10.

(name)? Does there exist a saved file with the specified name? If not, print an error message; otherwise, no message. Errors: 4, 10.

(name)/ Delete the saved file with the specified name, whether or not it is protected. Errors: 4, 5, 6, 8, 10.

(lun1)=(lun2) Unequip (lun1) and equip it to (lun2). Errors: 3.

(lun1)=(lun2)+	Unequip (lun1), equip it to (lun2), and protect it. Errors: 3, 7.
(lun1)=(lun2)-	Unequip (lun1), equip it to (lun2), and remove protection. Errors: 3, 9.
(lun)=(name)	Unequip (lun) and equip it to (name). If (name) is not a hardware type and there is no saved file with the specified name, create an empty saved file with this name, equipped to (lun). Errors: 4, 5.
(lun)=(name)+	Same as (lun)=(name), but protect the file after equipping it to (lun). Errors: 4, 5, 7.
(lun)=(name)-	Same as (lun)=(name), but remove protection after equipping. Errors: 4, 5, 8.
(lun)=(name)$	Save (lun) under the specified name. Errors: 3, 5, 7, 11, 12, 13.
(lun)=(name)?	Unequip (lun) and equip it to (name), but do not create a saved file if none exists under this name. Errors: 4, 10.
(name1)/(name2)	Change the name of file named (name1), so it has (name2). Errors: 4, 5, 6, 8, 10, 12, 13.
(name1)/(name2)+	Change name of file (name1) to (name2), and protect it. Errors: 4, 5, 6, 8, 10, 12, 13.
(name1)/(name2)-	Change name of file (name1) to (name2), and remove protection. Errors: 4, 5, 6, 8, 10, 12, 13.

A *DEFINE statement may have several arguments. If an error occurs in one of them, an error message is printed and DEFINE goes to process the other arguments. There may be several error messages from one *DEFINE statement.

Usage of *DEFINE statements.

DEFINE may be used in any of the three modes of OS-3:

1. Batch: To use DEFINE in a batch job, punch a 7 and 8 in column 1, then *DEFINE, a comma, and the arguments. All of the characters needed are available on the key punch except the question mark. To represent a question mark, punch 0,7,8 in a single column. (This will print on the line printer as an "and" symbol ∧.) The first 72 columns of the card may be used.
2. Teletype: Make sure you are in control mode (# printed out). Then type *DEFINE, a comma, the arguments, then press the RETURN key. Typing errors may be corrected by using the reverse slant (\), except that *DEFINE must be spelled correctly. A *DEFINE statement at a teletype should not be more than 80 characters long.
3. Display console: Make sure you are in control mode (≠ in upper left corner). Then type *DEFINE, a comma, the arguments, then press the SEND key. Use the "and" symbol (∧) in place of a question mark. The *DEFINE statement should not be more than 52 characters long. If no errors occur, DEFINE will blank the screen (≠ in corner) when finished. If there are errors, an error message will appear on the screen. The user should read the message, then press CLEAR and SEND (in this order). If there are more errors, another error message will appear. Repeat this procedure until the screen blanks.

A *DEFINE statement calls a program (DEFINE); hence, it is not possible to resume execution of another program that has been interrupted. One should therefore use *DEFINE statements only "between" other programs.

DIRECT and the directory feature.

As mentioned earlier, one can use DEFINE and DIRECT to maintain a saved file that contains the names of other saved files. To use this feature, one must have a saved file named DIRECTRY. One can create such a file by the statement:

*DEFINE,DIRECTRY

DEFINE will create a saved file named DIRECTRY and write a record on it containing the name DIRECTRY. Subsequently, whenever one uses a *DEFINE statement that refers to saved files, DEFINE will write information in the user's DIRECTRY file, indicating the names of saved files that exist or that do not exist in the OS-3 file system. DEFINE doesn't try very hard to do this; if the user does not have a file named DIRECTRY, or if his DIRECTRY is busy or protected, DEFINE drops the idea.

Some of the software available to OS-3 users will also write information in a user's DIRECTRY file if he has one. DECKEDIT (see separate publication) does this. The new version of EDIT will also do this. However, as of this writing, the OS-3 control statements (such as SAVE, DELETE, EQUIP, DESTROY) do not update a user's DIRECTRY, and there are no current plans to have them do this. Hence, if one uses OS-3 control statements SAVE, DELETE, or DESTROY to save or to delete files, one should then use a *DEFINE statement to get the information entered in his DIRECTRY. To do this, use a statement such as:

```
*DEFINE,ABC?,*QZORK?,D143A?
```

Using the question marks after the names causes DEFINE to try to discover whether the files exist or not. If they do, DEFINE writes information to this effect in the user's DIRECTRY. If a file does not exist, DEFINE prints an error message (such as *QZORK DOES NOT EXIST) and writes information in the user's DIRECTRY, indicating that this file does not exist.

The DIRECT program has two basic purposes: (1) to process the information in the user's DIRECTRY, and (2) to print a list of file names for the user. DIRECT reads the user's DIRECTRY file and and constructs a table of file names, entering or removing names according to information written by DEFINE, DECKEDIT, and EDIT. Then it writes this table back out on the user's DIRECTRY file. Finally, DIRECT prints a list of the file names, together with the most

recent date and time that each file was referenced by one of the programs mentioned above. The DIRECT program is called by a control statement of one of the two forms shown below:

*DIRECT	Prints directory on unit 61.
*DIRECT,(lun)	Prints directory on the specified logical unit (0 to 100).

DIRECT may be used in a batch job, or from a teletype or display console. If it is used at a display console, and the output unit is the display console itself, DIRECT will display the list of file names, up to 19 of them at a time. When the user presses SEND, he will get some more names, if there are any more.

By the time DIRECT begins to print the directory, it has already done all of its essential work. So, if a user stops DIRECT after it starts printing, nothing will be lost except the output to the user's teletype or display console.

If one uses *DEFINE to equip public files that belong to someone else, DEFINE will enter the names of such files in the user's DIRECTRY, since DEFINE does not "know" that the files don't belong to the user. To remove such names from a DIRECTRY, use a *DEFINE statement, attempting to remove protection from them. For example:

```
*DEFINE,*LIB-
```

This causes DEFINE to attempt to remove protection from the file. DEFINE "discovers" that it cannot do so, because the file is public. DEFINE will then write an error message (such as *LIB IS PUBLIC) and also write information in the user's DIRECTRY to cause the name to be removed.

Error messages from DIRECT.

There are three different error messages that one may get when calling *DIRECT.

DIRECTRY DOES NOT EXIST	means that the user does not have a file named DIRECTRY.
DIRECTRY IS EMPTY	means that there are no file names in the user's DIRECTRY.

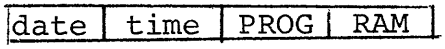
DIRECTRY IS BUSY means that someone else is using the user's DIRECTRY (and, of course, the same job/user number).

Format of information in DIRECTRY.

This section describes the format of the information that is written in a user's DIRECTRY. It is intended for the use of those who may wish to code their own programs to maintain DIRECTRY's, and persons who only wish to use DEFINE and DIRECT need not be concerned with this section.

The information in a DIRECTRY consists of a sequence of BCD records, each of them 4 words or longer. There should not be any file marks in the DIRECTRY. This makes it possible for a program (such as DEFINE) to equip the user's DIRECTRY, "search forward past file mark" to get to the end of the file, then write one or more records on the end of the file. The program which is updating a DIRECTRY should unequip it when finished.

All records written in a DIRECTRY contain a date and time in the first two words. The date is in the first word, obtained from register 37_8, and the second word contains the time, from register 22_8. The rest of the record contains one or more file names to be entered or removed. If a name is to be entered, one simply writes two words containing the name. (Both words must be present, even if the second one is all blanks.) If a name is to be removed, three words are used; the first word contains +0, and the next two words contain the name to be removed. (This convention relies on the fact that a valid OS-3 file name cannot have zeros for its first four characters.) Several names may be entered and/or removed with one record. Here are some examples:



Enters name PROGRAM in DIRECTRY.

date	time	+0	DATA	5		
date	time	ZORC	H	+0	*ZAP	

Removes name DATA5.

Enters ZORCH, removes *ZAP.

The maximum length allowed for a record in a DIRECTRY is fairly large; up to 1000 words or so is okay.