A Beginner's Guide to UDC's and JCW's: How to Use Them to Your Benefit

David L. Largent
The N.G. Gilbert Corporation
P.O. Box 1032
700 South Council
Muncie, IN 47308-1032

1.0 Introduction - "Is this for me?"

How many times have you forgotten to issue a FILE command before running a program? How often have you typed a FILE command wrong? Do you ever get tired of typing the DELETESPOOLFILE command? Have you been looking for a way to make your job streams a little "smarter"? Would a way to automate some of your procedures be helpful? If you have any of these problems and have never considered using UDC's and/or JCW's, now is the time to do so.

Over the course of the following pages will be found answers to questions related to User Defined Commands (UDC's) and Job Control Words (JCW's) such as:

- What are they?
- How can they help me?
- How are they created?
- How are they used?
- What are some common problems that arise?
- Are they worth the effort of learning something new?

To find these answers, a number of example UDC's, job streams, and programs will be examined to see how and where UDC's and JCW's can be used and how they work.

This paper is directed at new users of an HP3000 who have not yet explored the world of UDC's and/or JCW's. However, any user may find some tidbits to put in their "toolbox" that will prove useful either now or in the future. This paper assumes the reader has a general knowledge of the HP3000; specifically, the following are "prerequisites" for this paper:

- Knowledge of the HP3000 accounting structure (i.e., File/Group/Account/User).
- Knowledge of how to use an ASCII text EDITOR (e.g., EDITOR/3000).
- Knowledge of many MPE commands and how to use them.
- Knowledge of job streams and how to use them.

One further note. The information provided in this paper is correct and up-to-date (to the best of my knowledge) and reflects the way UDC's and JCW's are used and work as of the G.02.04 version of MPE V/E (UB Delta 4). I am aware that changes have been made in MPE XL, but have not made any attempt to include that information in this paper.

For those of you who have chosen to stay with me, here is what you can expect. First we will examine User Defined Commands (UDC's); second, Job Control Words (JCW's) will be examined; third, we will take a look at how UDC's and JCW's can be used together; and finally, we will take a look at what we have learned and decide if it is worth the effort to put these powerful new tools to work. So. . .

2.0 "What are User Defined Commands (UDC's)?"

UDC is one of many acronyms used in the HP3000 world. This one comes from the phrase User Defined Command. A UDC is a command that is designed for some user's convenience. It is made up of standard MPE commands and/or other UDC's. A UDC can replace a single, long command or may replace a long sequence of many commands. They provide a short cut (a more precise way!?!) to accomplish some particular task. By entering the UDC name, the predefined commands are automatically executed. Think of them as just another MPE command.

2.1 "How can UDC's help me?"

A UDC may be used in most any situation where a standard MPE command may be used, even within another UDC. They can be used in both sessions and job streams. They cannot, however, be executed by using the COMMAND intrinsic or from within subsystems unless you are in BREAK. With that as background information, how can they help you?

UDC's are perfect solutions for many "problem" situations. A long sequence of MPE commands can be replaced by a single UDC name, thus decreasing the time it takes to accomplish the task (less typing), and also eliminating typing errors. A lengthy command like DELETESPOOLFILE can be replaced by a UDC name such as RPUR. A complicated sequence of commands, such as a long list of FILE commands before running a program, can be reduced to one UDC name such as PAY. Many of your boring, repetitive tasks can be automated by using UDC's. UDC's can also be used to protect your system from both the naive user and the too-knowledgeable user. They can also cause something to automatically happen at log on time. These all serve as examples of ways to make the HP3000 more "user-friendly".

By this point in time, you probably have half a dozen UDC ideas in your head just waiting to get out. So. . .

2.2 "How do I create a UDC?"

First of all, slow down! UDC's, as with most everything in life, are best created with PPP -- proper prior planning. If thought is given to each UDC you create, a lot of changes can be avoided later and you will likely end up with a more functional set of UDC's that are easier to use. For some people, creating a good UDC is an art, rather than a science -- they may put as much thought and care into a UDC as they put into developing a program.

Now that you have slowed down a bit (you have haven't you?!!), give some thought to what MPE commands get used often on your system. Are there particular tasks that always require the same sequence of commands? Are there commands or tasks that may not be difficult for us programmers to master, but may be intimidating for the user? Now take your list (you did write them down didn't you?) and evaluate each command or task. Some of them may be used so seldom that it is not worthwhile to create a UDC. (That can also be a reason to create a UDC for a task, so that the sequence of steps need not be remembered!) Some of the commands may be so short that creating a UDC for them would only save two or three key strokes. You must make a judgement call for each command or task as to whether the convenience of having the UDC available for use is worth the time and effort of creating it.

2.21 Defining the UDC.

Let us assume that you have chosen to create a UDC for the SHOWJOB command in its simplest form. You would like to be able to type the letter "J", and have the system behave as if you had typed "SHOWJOB". A UDC to do this could look like:

```
J                    Header section
OPTION LIST, HELP    Execution options section
SHOWJOB              Body section
**                   UDC separator section
```

As shown above, every UDC consists of four main sections: a header section, an optional execution options section, a body section, and a UDC separator section. The first line in this example is the header section, containing the actual command the user will type. The second line provides some execution options to the system. The third line makes up the body section, containing the actual command to be executed. The fourth line serves as a separator between UDC definitions in a UDC file.

2.211 The Header Section.

The Header Section of a UDC consists of the command name, parameters, and their optional default values. A UDC header may extend over more than one physical line or up to a maximum of 320 characters. Each line to be continued must have as its last nonblank character an ampersand (&).

The command name is what the user will actually type (along with any needed parameter values) to cause the UDC to be executed. It is composed of a maximum of sixteen alphabetic or numeric characters and must begin with an alphabetic character. The characters "RFA" may not be a UDC name nor may they be the first three characters of a UDC name, as this has been reserved by HP for internal use only. Use common sense when naming your UDC's. Use a derivative of the actual MPE command(s), or if naming a UDC for a particular task, give it a name that is descriptive of the task. It is a good idea to avoid single letter UDC names to reduce the chance of executing the command by mistake (at least for any that may be "destructive").

Parameters are variables that are assigned either a default value or a value provided by the user along with the command name. These variables are specified on the same line as the command name and may be used in the body of the UDC to make the UDC more flexible and less specific. You may have a maximum of sixteen parameters in one UDC.

Each parameter must have a name. A parameter name must begin with an alphabetic character; the remainder may be alphabetic or numeric characters. The maximum length for a parameter name is seventy characters. However, the actual maximum may be less -- a parameter name may not be split between physical lines. The same is true for default values for parameters. Further constraints on the length of the parameter name may apply because of the position where the parameter is used in the body section.

If a UDC parameter does not have a default value specified, then it is considered a required parameter and one must be provided when the user executes the UDC. If a value for a required parameter is not provided by the user, an appropriate error message is displayed and the UDC is not executed.

Let us change the requirements for our J UDC. We now want it to do one of three things:

```
SHOWJOB JOB=@     (show all jobs)
SHOWJOB JOB=@S    (show sessions only)
SHOWJOB JOB=@J    (show batch jobs only)
```

This can be accomplished with one UDC if we use a parameter. The header section will now look like:

```
J WHAT2SHOW = " "
```

The command name is still J. The character string "WHAT2SHOW" is the parameter name. A default value of " " has been specified. This means that if a user just types "J", a list of all jobs will be displayed. Alternatively, if the user types "J S" the list will show sessions only.

As can be seen in the example above, a default value for a parameter is specified by placing an equal sign (=) after the parameter name, and then following that with the default value. If the default value contains spaces or special characters, then it will need to be enclosed in quotation marks ("). If no spaces or special characters are needed in the default value, the quotation marks are optional. When a default value is not provided for a parameter, the only thing that appears in the header section is the parameter name.

If more than one parameter is used in a UDC, default values may be specified for all, some, or none of them. That is, some of the parameters can be required while others are optional. A comma (,) is used to separate a parameter name from the previous parameter name (or default value if one is specified).

From the user's point of view, there are two ways UDC parameters can be thought of: keyword and positional. When a user is executing a UDC, either approach may be used, but not both at the same time. A keyword parameter is one in which the parameter name is typed, followed by an equal sign (=), followed by the parameter value. In this way, values for parameters may be provided in any order. For example:

```
J WHAT2SHOW="S"
```

is the way the user would execute the J UDC using a keyword parameter. A positional parameter is one in which the value is specified for each parameter in the order they were defined in the UDC. If a new value is not provided for an optional parameter, its position must still be "held" by a comma (,). The J UDC executed with a positional parameter would look like:

```
J S
```

The concept of keyword and positional parameters is the same as for the standard MPE commands, except that only one approach may be used on a given execution of a UDC.

When providing numeric values for parameters (either the default values or the user's actual values), both decimal and octal numbers may be used. Octal numbers are indicated by preceding them with a percent sign (%). As you would expect, if the user provides a value for a parameter that has a default value, the user's value is the one that will be used. If the parameter value a user needs to provide contains spaces or special characters, that value will need to be enclosed in quotation marks (").

2.212 The Execution Options Section.

The execution options section of a UDC consists of choices from each of four pairs of options. Each of these four pairs has a

default and, therefore, if the defaults are what is desired, this entire section can be left out. These options control the operation and use of the UDC. If the execution options section is provided, it must appear as the next line following the header section and will consist of a single line. The line must start with the word OPTION. The rest of the line contains the option or options you have chosen with a comma (,) between each one if you list more than one. The four option pairs are discussed in the following paragraphs.

LIST/NOLIST. The default is NOLIST. The LIST option will cause the text of the body section to be listed on the standard list device with the parameter values substituted as each line is executed. The NOLIST option will not list any of the text of the body section. It is a good idea to use the LIST option on UDC's that are replacements for standard MPE commands. This way the actual command is before you and you will be less likely to forget what the MPE command is when you need to use it on another system that doesn't have your UDC! The NOLIST option is generally a good choice for UDC's that are implementing user tasks.

BREAK/NOBREAK. The default is BREAK. If the NOBREAK option is chosen, the commands that make up the body of the UDC will be nonBREAKable. That is, pressing the BREAK key will not cause the command or program to stop. If the BREAK option is chosen, the commands that make up the body of the UDC will be BREAKable (if they are normally BREAKable as MPE commands). It is a good idea to use the NOBREAK option on any UDC that runs a VPLUS application. This eliminates the problems that occur when BREAK is pressed while in block mode. The use of the NOBREAK option will prohibit certain users from using any MPE commands and can serve as a way to discourage simple security breaches.

If the BREAK key is pressed during the execution of a UDC that is using the BREAK option, the following happens:

- If it was executing a nonprogram command, the UDC terminates.

- If it was executing a program, and the RESUME command is used, the UDC will also resume execution. An exception to this occurs if a SETCATALOG command is executed while in BREAK. In this case, the program is resumed but the remainder of the UDC will not be executed.

For you programmers, the NOBREAK option overrides the CAUSEBREAK intrinsic. A program containing CAUSEBREAK will not BREAK if it is executed from a UDC that has the NOBREAK option specified.

LOGON/NOLOGON. The default is NOLOGON. If the LOGON option is chosen, the commands specified in the body of the UDC will

automatically be executed when the user logs on. Only one LOGON UDC at the user level will be executed. If more than one LOGON UDC exists for the user level, only the first one will execute. The LOGON option can be put to good use on user task UDC's. By using this option, the user can logon and automatically be put into a menu or application program. If the NOBREAK option is also used, you can keep a user from gaining direct access to MPE commands. An example of this will be shown later.

HELP/NOHELP. The default is HELP. If the HELP option is chosen, you may type HELP followed by the UDC name and see the definition of the UDC. If the NOHELP option is chosen, this possibility does not exist. More will be said later about the use of the MPE HELP subsystem. It is good to choose the HELP option unless you have a security sensitive situation. If you specify both the NOLIST and NOHELP options and an error occurs during execution of the UDC, the error will be reported but the line containing the error will not be listed.

2.213 The Body Section.

The body section of a UDC consists of one or more MPE commands and/or UDC's which will be executed when the user types the UDC name. Other noncommand text such as data for subsystems or application programs may not be included. A logical line of the body may extend over more than one physical line for up to a maximum of 320 characters. Each line to be continued must have as its last nonblank character an ampersand (&). The body section follows the execution options section, if any options have been listed. If the execution options section is not present, then the body section will follow the header section.

There are a few restrictions of which to be aware. The REDO command may not be used in a UDC. The DATA, JOB, and HELLO commands may be used, however, they will cause the current job or session to logoff and effectively terminate execution of the UDC; no new job or session will be initiated.

Let us go back to our last version of the J UDC. We wanted it to do one of three things:

```
SHOWJOB JOB=@
SHOWJOB JOB=@S
SHOWJOB JOB=@J
```

We last defined the header section as:

```
J WHAT2SHOW = " "
```

The body section needed to satisfy our requirements will consist of a single MPE command:

```
SHOWJOB JOB=@!WHAT2SHOW
```

In this example, all the characters preceding the exclamation point (!) will be left as shown. The character string "!WHAT2SHOW" will be replaced by either the default value of " ", or whatever the user types as a parameter value following "J". This revised command image is then what is sent to the MPE command interpreter for execution. Thus, if the user types:

```
J S
```

the command image that will be executed will be:

```
SHOWJOB JOB=@S
```

Alternatively, if the user just types:

```
J
```

and uses the default parameter value, the command image that will be executed will be:

```
SHOWJOB JOB=@
```

As is shown above, placing an exclamation point (!) immediately before a parameter name in the body section of a UDC will cause MPE to substitute the value of that parameter into that position of the command. If a parameter name does not follow an exclamation point, then an error will be reported to the user. If an exclamation point does not precede a parameter name, no substitution will be attempted, and the parameter name will be left in the command.

Normally, an exclamation point signifies a parameter name. If you want to use the exclamation point but not have MPE try to substitute a value, use an even number of exclamation points. An odd number causes MPE to attempt a substitution. Each pair of exclamation points will be translated into a single exclamation point. For example, consider the following UDC:

```
EXAMPLE PARM
OPTION LIST
COMMENT !!PARM
COMMENT !!!!!PARM
**
```

If the user was to execute it by typing

```
EXAMPLE TEST
```

the two comments would be displayed as:

```
COMMENT !PARM
COMMENT !!TEST
```

Sometimes you will need to insert a parameter into the middle of a character string in the UDC body section. As an example, the following UDC is presented:

```
PREPS PROGNAME, MAXDATASIZE=15000
OPTION LIST
PURGE !"PROGNAME"X
PREP !"PROGNAME"U,!"PROGNAME"X;MAXDATA=!MAXDATASIZE;RL=PRRL
SAVE !"PROGNAME"X
**
```

In this UDC, two parameters are used: PROGNAME and MAXDATASIZE. MAXDATASIZE has a default value of 15000. When the UDC is executed, it will purge a file, prepare a file, and then save the program file just created. The parameter name PROGNAME is enclosed in quotation marks (") to separate it from the letters "X" and "U" that follow. Without the quotation marks, there would appear to be invalid parameter names following the exclamation points. This is only a concern if the character immediately following a parameter name is a letter or number.

If the UDC was to be executed by typing:

```
PREPS PAY
```

the actual command images executed would be:

```
PURGE PAYX
PREP PAYU,PAYX;MAXDATA=15000;RL=PRRL
SAVE PAYX
```

The parameter value "PAY" corresponds to parameter PROGNAME because they are both the first item provided in the list. Since a second parameter value was not provided when the UDC was executed, the default value for MAXDATASIZE was used.

If an error is encountered as the commands of a UDC are executed, an error message will be displayed and the UDC will terminate. If the UDC is being executed in a job stream, the job stream will also terminate. Sometimes it is desirable for the UDC to continue with the next command regardless of whether the previous command resulted in an error. To accomplish this, a CONTINUE command must be inserted immediately preceding the command that may result in an error. Consider the following UDC:

```
REPORTS
OPTION LIST
CONTINUE
RUN PROG1X
RUN PROG2X
SHOWTIME
**
```

In this example, two programs are run and then the time of day is displayed. If program PROG2X results in an error (i.e., it aborts) when it is run, the remainder of the UDC will not be executed. That is, if program PROG2X aborts, nothing else will be done. Program PROG2X will always be run regardless of whether program PROG1X results in an error. The CONTINUE command tells MPE to continue with the next command, even if the current command fails. This "override" capability only applies to the command immediately following the CONTINUE command.

UDC's can perform tasks much more complex than those already covered by using sophisticated command sequences and/or referencing other UDC's from within the body section of the UDC. By using control characters and escape sequences, you can control the appearance and location of information as it is displayed by a UDC. The terminal's function keys can be "loaded" by a UDC, thus making your terminal a little more "user-friendly". Examples of these will be provided later.

As mentioned earlier, a UDC may reference or execute another UDC. This process is referred to as nesting. There is a limitation however: a UDC may only reference another UDC that is defined later in the UDC file (we will discuss UDC files in a few more paragraphs). As long as the reference occurs before the definition, UDC's can be written in a structured way.

Consider the following UDC's:

```
INFO
OPTION LIST
ME
J
T
**
J
OPTION LIST
SHOWJOB
**
ME
OPTION LIST
SHOWME
**
T
OPTION LIST
SHOWTIME
**
```

In this example, we have defined four UDC's: INFO, J, ME, and T. The INFO UDC will cause the J, ME, and T UDC's to be executed one after the other in the order listed in the INFO UDC. When the J UDC is executed, it will execute the SHOWJOB command; the ME UDC will execute the SHOWME command; and the T UDC will execute the

SHOWTIME command. So, by simply typing "INFO", the user will be provided information from all three of the MPE commands automatically.

2.214 The UDC Separator Section.

The last section of a UDC definition is the UDC separator. This consists of a single line that must start with an asterisk (*) in column one. The rest of the line is yours to do with as you wish. The purpose of this section is to separate one UDC from the next one in the UDC file and must be the last line of the UDC definition.

So, there you have it in ten words or less (give or take a few thousand). You now know how to define a UDC. But once you have a number of UDC's defined, what do you do with them? The answer to that question is. . .

2.22 Put Your UDC Definitions in a UDC File.

A UDC file is simply an ASCII text file that contains one or more UDC definitions. Any ASCII text editor may be used to create a UDC file. The record length of the UDC file may be any length; however, 72 character records is what MPE is expecting. If you set your record length less than 72, your UDC's likely will not execute properly. Using a record length larger than 72 characters will work; however, only the first 72 characters of each line will be read by MPE. A UDC file may contain any number of UDC definitions.

The UDC's should be entered in some logical sequence to make maintenance easier -- my suggestion is alphabetically by the UDC name. An exception: remember that if you are using nested UDC's, the definition of a UDC must be after its reference(s). So, either name them appropriately or include them in the file out of sequence. If you are using the escape character in your UDC's, an easy way to key it in is to type in some otherwise unused character (e.g., the ^ character) in place of the escape character. Then when you are finished entering the UDC definitions, globally change all occurrences of that character to the escape character. The reverse sequence can be used if you need to modify an existing UDC file. Just remember to change them back before you keep the file.

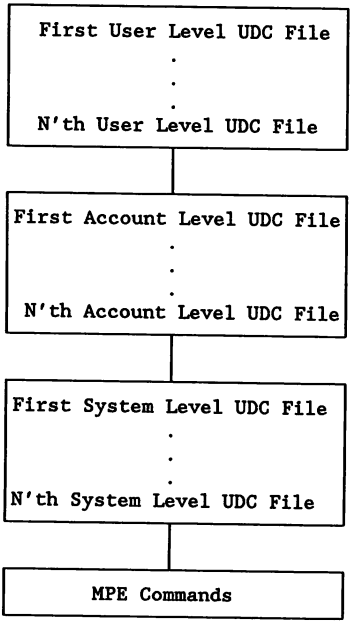
Once you have entered all of the UDC definitions using your text editor, save the file with some meaningful name. My preference is to use the letters "UDC" as the last three characters of the file name. Doing so makes it easy to locate UDC files with the LISTF command. You may keep your file with or without line numbers -- MPE does not care. You may create multiple UDC files; however, for performance reasons (which we will discuss later), it is best to limit the number of UDC files that exist for one user. If you wish, a lockword may be assigned to the UDC file.

Simply creating a UDC file with a text editor does not yet make the UDC's available for use. The next step is to tell MPE that you have a UDC file you wish to use. However, before we discuss that, let us take a look at account and system level UDC's.

2.23 Account and System Level UDC's.

Up to this point, we have been discussing UDC's in general. There are actually three different levels of UDC's: user, account, and system. A UDC file may be set up by a user for his/her own use. An account manager may create a set of UDC's for use by all users of that account. The system manager may create a set of UDC's for use by all users on the system. So now you have one more item to consider when you are creating a UDC: Who should have access to it?

When the MPE command interpreter is given a command (either an MPE command or a UDC), it follows a predefined hierarchy to decide what to do with the command in question. Shown below is that hierarchy:



This hierarchy is followed regardless of whether it is trying to locate a command provided at a colon (:) prompt or a command that is coming from a UDC. This is the reason that when using nested

UDC's, the UDC definition must follow the reference(s) to that UDC.

There is an exception to this hierarchy: UDC's with LOGON listed as an option. When a user logs on, one logon UDC, at most, will be performed at each UDC level. The system level logon UDC (if present) will execute first, then the account level logon UDC (if present), and finally, the user level logon UDC (if present) will be executed. If more than one logon UDC exists at a level, only the first logon UDC at that level will be executed but will not affect the execution of other levels' logon UDC's.

Another implication of this hierarchy is that when a UDC at one level has an identical name to a UDC or command at another level, the following rules apply:

1. A UDC in user level UDC file 1 takes precedence over user level UDC file n.

2. A UDC in user level UDC file n takes precedence over account level UDC file 1.

3. A UDC in account level UDC file 1 takes precedence over account level UDC file n.

4. A UDC in account level UDC file n takes precedence over system level UDC file 1.

5. A UDC in system level UDC file 1 takes precedence over system level UDC file n.

6. A UDC in system level UDC file n takes precedence over the MPE commands.

What all of this means is that you can set up a UDC with exactly the same name as an MPE command (or UDC) and effectively redefine that command. For instance, if you wanted to keep a user from using the PURGE command, the following UDC could be established:

```
PURGE FILENAME = "$NULL"
OPTION LIST
COMMENT YOU HAVE NOT BEEN GIVEN ACCESS TO THIS COMMAND.
**
```

Unless you are specifically taking advantage of this capability, it is best to give all of your UDC's unique names. Here is another example of these rules and the hierarchy: if both a system level and a user level UDC have the same name and an account level UDC references a UDC by that name, then the system level UDC will be executed rather than the user level UDC because the system level UDC is "after" the account level UDC that referenced it, where as the user level UDC would be "before" that reference.

A user must have special capabilities to establish account or system level UDC's. To establish account level UDC's, the user must have the AM capability; to establish system level UDC's, the user must have the SM capability. Additionally, if system level UDC's are established, the system level UDC files must either have their access security released or have the access security such that all users will have READ and LOCK access to the file(s). (For security reasons, the latter is preferred.)

2.24 Telling the System About Your UDC File(s).

As stated earlier, simply creating a UDC file with a text editor does not yet make the UDC's available for use. The next step is to tell MPE that you have a UDC file you wish to use. The MPE command to do this is SETCATALOG. The simplest syntax of the SETCATALOG command is:

```
SETCATALOG udcfilename[/lockword]
```

where "udcfilename" is your UDC file name and "lockword" is an optional lockword assigned to the UDC file. This version will enable a UDC file for the user level only. The user executing the SETCATALOG command must have READ and LOCK access to the UDC file. Unless you have account or system manager capabilities, you must logon as the user that you want to enable the UDC file for. The SETCATALOG command may be issued from a session, job, or in BREAK. It may not be issued from a program, nor is it BREAKable.

When SETCATALOG is executed with a file name provided, three things occur. The command interpreter searches the UDC file for errors: problems like specifying an option that is not a valid option are caught; problems like syntax errors in the body section of the UDC are not caught. If no errors are found in a UDC, then an entry is established in a directory that will eventually contain entries for all UDC's in the UDC file. The UDC file name and optional lockword is stored in a system catalog of all UDC users. This system catalog is file COMMAND.PUB.SYS. The directory is stored in an extra data segment for that session or job. Since the lockword (if one is provided) is stored in COMMAND.PUB.SYS, a user does not need to know the lockword to logon or to access the UDC. In addition to the UDC file name (and lockword), the user name and account and the UDC level (user, account, system) are also stored in COMMAND.PUB.SYS.

As you could probably guess, the file COMMAND.PUB.SYS must exist before any SETCATALOG is attempted. If the file does not exist, the system manager or supervisor must build it. The file should be built with a record size of twenty words. The maximum number of records for the file can be determined by calculating the sum of:

1 for each user.account that will have user level UDC's; plus
1 for each user level UDC file to be used; plus

1 for each account that will have account level UDC's; plus
1 for each account level UDC file to be used; plus
1 if system level UDC's are to be used; plus
1 for each system level UDC file to be used.

So, the BUILD command used by the system manager might look like:

```
BUILD COMMAND.PUB;REC=-20,32;DISC=500
```

To secure this file so that all users may utilize UDC's, but only the system manager may read or modify it (since there could be sensitive information stored there), the system manager should enter:

```
ALTSEC COMMAND.PUB;(X:ANY;R,L,W,A:CR)
```

Taking this last step will provide execute access to all users (which is all they need), and restrict all other access to the creator of the file (in this instance, the system manager).

If you have more than one UDC file that you wish to enable at the user level, the syntax of the SETCATALOG command becomes:

```
SETCATALOG udcfilename1[/lockword],udcfilename2[/lockword]...
```

The UDC file names are listed (with the optional lockwords) with a comma (,) inserted between each one. The position of a UDC in the directory determines which other UDC's it may reference (remember the hierarchy discussed a few paragraphs back?). The UDC files are opened and scanned in the order they appeared in the last SETCATALOG command for that UDC level. All user level UDC's are entered into the directory first, followed by all account level UDC's, and finally, all system level UDC's.

So, now you know how to "turn on" or enable user level UDC's. What if you want to disable or "turn off" user level UDC's? To accomplish this, you again use the SETCATALOG command, except this time do not provide a UDC file name:

```
SETCATALOG
```

When a UDC file name is not provided with the SETCATALOG command, all entries and references to any user level UDC files in the UDC directory and COMMAND.PUB.SYS that are currently enabled for the user issuing the command are eliminated. The UDC files themselves, however, are not purged. To re-enable your UDC files, you would simply type the SETCATALOG command, followed by your UDC file name(s) again. Disabling user level UDC's does not have an immediate affect on other job or sessions that are still logged on with the same user name. They may continue to use the disabled UDC's until they logoff. Conversely, if you enable some UDC's, other users will not have access to them until they logon again.

Now, suppose sometime previously you have typed:

```
SETCATALOG UDC1,UDC2,UDC3
```

You, therefore, have the UDC's from three UDC files enabled (plus any account or system level UDC's!). For whatever reason, you no longer need the UDC's in file UDC2. To eliminate them, but still keep the others, you need to type:

```
SETCATALOG UDC1,UDC3
```

Anytime the SETCATALOG command is executed, it has the effect of removing all user level UDC's for the user entering the command. Additionally, if a UDC file name(s) is provided, those UDC's are re-cataloged and available for use.

At some point in time (probably very soon after you start using UDC's), you will need to modify or add to an existing UDC file. There is not much more involved than simply using a text editor to make the changes or additions; however, you do need to know about a "gotcha". Because of the way UDC's are enabled, whenever a user is logged on to the system, any UDC file(s) associated with that user (including account and system level UDC files) are considered "open". What this means is that you may not alter that UDC file in any way as long as the user is logged on and has the UDC file enabled. You may not PURGE, RENAME, or (in EDITOR) modify and KEEP that UDC file. Any attempt to alter the file will result in the error message "EXCLUSIVE VIOLATION".

So, how do you ever make changes or additions?!? You have five options (take your pick -- they all work well under different situations):

1. Have all users (and jobs) that are using the UDC file log off (except you of course). You use the SETCATALOG command to disable the UDC file in question. Make your changes to the file, and keep it. Re-enable the UDC file by using the SETCATALOG command. Notify the users they may log back on.

2. Have all affected users (including yourself) disable the UDC file by using the SETCATALOG command. Make your changes to the file, and keep it. Have all users re-enable the UDC file by using the SETCATALOG command - or - you use the SETCATALOG command and have the other affected users logon again. Watch out for batch jobs when using this option.

3. Have all affected users (and jobs) logoff the system. You logon so that you can modify and keep the UDC file, but as a user that does not have this UDC file enabled. Make your changes to the file, and keep it. Notify the users they may log back on.

4. Logon so that you can modify and keep the UDC file. Make your changes to the file and keep it under a new file name. Later, after all affected users and jobs have logged off, disable the UDC file, PURGE the old UDC file, RENAME the new file to the old file, and re-enable the UDC file with the SETCATALOG command.

5. Any combination or variation of the above four options that you can get to work.

It is important to understand that all affected users must take some action (logoff/on or SETCATALOG) because the SETCATALOG command only takes effect for the session or job that issues it and for future logons. It does not affect other jobs or sessions currently logged on.

You now know everything (well almost everything) I know about using the SETCATALOG command with user level UDC files. But what about account and system level UDC files? How do they get enabled and disabled for use? The answer again is the SETCATALOG command! We add one more parameter to the previous syntax:

```
SETCATALOG udcfilename1[/lockword][,udcfilename2/[lockword]]...;ACCOUNT
SETCATALOG udcfilename1[/lockword][,udcfilename2/[lockword]]...;SYSTEM
```

The ACCOUNT parameter specifies that the UDC file(s) being enabled should be available for all users in the account. This parameter requires the account manager capability. Reading between the lines, you have probably realized that you must logon to the account that the UDC file is to be enabled for. For example, if you want to enable UDC file ACCTUDC for account XYZ, first logon as the account manager of account XYZ, then enter:

```
SETCATALOG ACCTUDC;ACCOUNT
```

If you wish to disable the account level UDC files, you would enter:

```
SETCATALOG;ACCOUNT
```

Again, you must have the account manager capability to use the ACCOUNT parameter. Taking this new information into account, all previous information pertaining to the SETCATALOG command and user level UDC files also applies to account level UDC files.

The SYSTEM parameter specifies that the UDC file(s) being enabled should be available for all users of the system. This parameter requires the system manager capability. For example, if you want to enable UDC file SYSUDC for all users of the system, just logon as the system manager, then enter:

```
SETCATALOG SYSUDC;SYSTEM
```

If you wish to disable the system level UDC files, you would enter:

```
SETCATALOG;SYSTEM
```

Again, you must have the system manager capability to use the SYSTEM parameter. Taking this new information into account, all previous information pertaining to the SETCATALOG command and user level UDC files also applies to system level UDC files.

If you have the account manager or system manager capability, you may use another parameter of the SETCATALOG command:

```
SETCATALOG udcfilename1[/lockword][,udcfilename2[/lockword]]...
           ;USER=user[.account]
```

where "user" is a user name and "account" is an account name. This allows you to enable a UDC file for a user other than the one you are logged on as. Account managers may enable UDC files for any user in their account. The system manager may enable UDC files for any user on the system. They will not take effect, however, until the next time the user logs on.

This parameter may also be used to disable UDC files for other users:

```
SETCATALOG;USER=user.[account]
```

The same restrictions and capabilities apply for account managers and the system manager, as were just discussed above.

One last parameter that you may wish to know about:

```
SETCATALOG ...any other parameters...;SHOW
```

This will list the UDC file names and UDC definitions in each of those files as they are scanned. This is helpful for locating where an error is occurring when you use the SETCATALOG command, but I don't suggest it for general use. It can take a while to list all of the UDC definitions in a large UDC file. Additionally, any account or system level UDC's are listed after the user level UDC's are scanned (remember -- they go into the user's directory too!).

2.25 What happens when I logon and have some UDC's enabled?

At logon time, any user that has one or more UDC file(s) enabled for his/her use (user, account, or system!) will cause a fair amount of CPU and disc utilization to occur. As a result of previous SETCATALOG's, the file COMMAND.PUB.SYS contains the UDC file names enabled for every user, as well as those enabled for each account and the system. At logon time, this file is searched to identify what UDC files are enabled for the user in question. As each one is identified, that UDC file is opened and the contents of it are read. As the UDC file is scanned, entries are created and placed in a

directory of UDC's for the user. That directory is created in an extra data segment for the session or job.

When you enter a command or UDC name, the command interpreter searches the UDC directory (the extra data segment) for that which you typed in, starting at the beginning of the directory. Remember that entries are placed in the UDC directory in user level, account level, system level sequence. If the command is found in the UDC directory, the UDC body for that command is read one line at a time and parameters are substituted into the line where appropriate. The command interpreter is then re-entered at a special internal entry point to interpret the new expanded command string and goes through the same steps just mentioned, except that this time, the UDC directory scan begins with the directory entry that follows the UDC currently being executed. (Now do you understand why the definition of a UDC must be after its reference?) If the command interpreter fails to find a match for a command string in the UDC directory, it then checks to see if it is a valid MPE command. This cycle is repeated until the end of the UDC definition that is being executed.

2.26 Looking at what UDC's are available.

There is a handy MPE command which will list all of the UDC names enabled for your session or job:

```
SHOWCATALOG [listfile]
```

where "listfile" is the name of the file (disc or printer) you wish the list to be sent to. Unless directed elsewhere with a prior FILE command, if you specify "listfile", the listing is sent to device class LP. If "listfile" is not provided, the list of UDC names will be displayed on $STDLIST (your terminal for a session). This command may be issued from a session, job, or in BREAK. It may not be issued from a program. The SHOWCATALOG command is BREAKable (it aborts execution of the command).

The output from the SHOWCATALOG command lists the UDC's currently enabled for your use, the level at which they are defined (user, account, system), and the file name in which they reside. A sample execution follows:

```
:SHOWCATALOG
UDC1.GROUP.ACCT
     AA                    USER
     BB                    USER
UDC2.GROUP.ACCT
     CC                    USER
ACCTUDC.PUB.ACCT
     DD                    ACCOUNT
```

```
SYSUDC.PUB.SYS
     EE                    SYSTEM
     FF                    SYSTEM
     GG                    SYSTEM
```

This example shows seven UDC's enabled from four different UDC files: three at the user level, one at the account level, and three more at the system level.

The SHOWCATALOG command has another parameter that can be useful, especially for account and system managers:

```
SHOWCATALOG [listfile];USER=user[.account]
```

where "user" is the name of a user, and "account" is an account name. This parameter permits you to specify a user other than yourself for which you want the SHOWCATALOG done. The output when this parameter is used will consist of only enabled UDC file names and which level they were defined at. No UDC names will be listed.

For example if you want to know what UDC file(s) are enabled for a user, you could type:

```
SHOWCATALOG;USER=DAVE
```

and the system would respond with:

```
USER UDC CATALOG FILE NAMES:
     UDC1.GROUP.ACCT
     UDC2.GROUP.ACCT

ACCOUNT UDC CATALOG FILE NAMES:
     ACCTUDC.PUB.ACCT

SYSTEM UDC CATALOG FILE NAMES:
     SYSUDC.PUB.SYS
```

There are a few limitations when using the "USER=" parameter:

1. A user with neither account nor system manager capabilities may only specify his/her own user name. A user may not obtain information for any other user.

2. A user with account manager capability may specify any user in his/her account. Additionally, if "@" is used for the user name, only the account level UDC file name(s) will be displayed.

3. A user with system manager capability may specify any user on the system. Additionally, if "@" is used for the user name and an account name is specified, the account level UDC file name(s) will be displayed. Also,

if "@" is specified for both the user and account, only the system level UDC file name(s) will be displayed.

A way of getting more information about a particular UDC is to use the MPE HELP command. Unless the NOHELP execution option was specified when the UDC was created, if you type

```
HELP udcname
```

where "udcname" is the name of a UDC, you will receive a listing of the UDC definition. For example, if you typed:

```
HELP J
```

(and the UDC we discussed sometime back was enabled), the system would respond with:

```
USER DEFINED COMMAND

J WHAT2SHOW=" "
OPTION LIST
SHOWJOB JOB=@!WHAT2SHOW
```

If the NOHELP execution option is specified when a UDC is created, and the HELP command is used for that UDC, the system will respond with:

```
Can't find anything under this command or in the table of contents.
```

unless the UDC name is the same as an MPE command, in which case you will receive information about the MPE command. It should be noted that you may not enter the HELP subsystem itself and get information about your UDC's. Only by using the HELP command as described earlier will you receive this information.

2.3 How about some examples?

Good Idea! I'll present some here; also see section four for examples of using JCW's and UDC's together. NOTE: Anytime you see the characters "<esc>" in this paper, read it as an escape character. That is, for purposes of this paper, I have used the characters "<esc>" in place of the escape character. However, if you were to type in the UDC, you would need to use the escape character instead. Additionally, the control character will be represented by the characters "<ctrl>".

We have been at this for a while, so let's take a look at some games -- I mean UDC's for a games user. I have a variation of the following UDC file set up on my system:

```
StartGames
Option LogOn
Display " <esc>&dA**************************************** "
Display " <esc>&dA*                                        * "
Display " <esc>&dA*    WELCOME TO THE HP 3000 GAME ROOM    * "
Display " <esc>&dA*                                        * "
Display " <esc>&dA**************************************** "
Display " "
Display " Type LIST to get a listing of the available games  "
Display " "
**
Games
Display " ADVENT.... (Adventure) A game of exploration.      "
Display " AMORT..... Creates amortization (loan) tables.     "
...continued for other games available...
**
Advent
Continue
Run ADVENT
**
Amort
Display "<esc>&dBNOTE: If you want the loan schedule printed on paper,  "
Display "<esc>&dB      wait for this phrase to appear...                 "
Display "<esc>&dB        ENTER THE LISTING DEVICE (RETURN FOR $STDLIST)?"
Display "<esc>&dB      then press the BREAK key. You will get a colon(:)"
Display "<esc>&dB      Type AMORTPRINT, and press the RETURN key. The    "
Display "<esc>&dB      screen will now say...                            "
Display "<esc>&dB        READ pending                                    "
Display "<esc>&dB      Now type LP, press the RETURN key, and go on to   "
Display "<esc>&dB      answer the other questions.                       "
Display "<esc>&dB                                                        "
Display "<esc>&dB      If you just want the schedule to print on the     "
Display "<esc>&dB      screen, press the RETURN key instead of doing all"
Display "<esc>&dB      of the above.                                     "
Continue
Run AMORT
Reset List
List
**
AmortPrint
File LIST=AMORTRPT;Forms=8 1/2 by 11 INCH PAPER REQUIRED <ctrl>G.
Resume
**
...continued for other games available...
Stop
Abort
**
List
Display "  Available Games...                                        "
Display "      ADVENT      AMORT      ANIMAL      BIOSIN            "
Display "      BIOSINP1    BJ         CHESS       FIVEROW           "
Display "      FOOTBALL    GRIC       LANDERP     OTHELLO           "
```

```
Display "       SAHARA       STARTREC  TREK2640   UBOAT          "
Display "       ULTIMA
Display " "
Display "  If you would like a one line description of each of"
Display "  the games, type GAMES, and press the return key.    "
Display " "
**
```

Items of interest in the example UDC file above:

1. The LOGON execution option is used so the welcome message appears automatically.

2. No, there is not an MPE command DISPLAY! Display is a system level UDC I have set up to cause the parameter value provided to be displayed by using the COMMENT command. (We will take a look at the UDC definition for DISPLAY later.

3. Escape sequences are used to control the appearance of text on the screen. In the case of STARTGAMES, it will blink, and in AMORT, the text will be shown in inverse video.

4. The CONTINUE command is used in all of the individual game UDC's to guarantee that the LIST UDC will always be executed, even if the program aborts.

5. The RESUME command is used in AMORTPRINT to automatically put them back in the program after they press the BREAK key.

6. Notice that no options were provided in most UDC's. Since the default is NOLIST, the command(s) will not be displayed on the screen. However, in UDC DISPLAY, OPTION LIST is used so that the messages passed to it will be displayed.

7. The LIST UDC must be at the end after all of the individual game UDC's because each one of them ends with a reference to LIST.

8. The STOP UDC exists to provide the user a "graceful" way to end a game they do not wish to continue. The users know to press the BREAK key and type STOP if they find themselves in this situation.

O.K. We have had our fun with the games. Time to get back to some serious work. The next series of examples come from some of my application systems. These UDC's are generally user level UDC's.

First, a few UDC's from the accounts payable system. In UDC file APINUDC, I have the following UDC:

```
Payable
Option LogOn,NoBreak
Continue
Payable
Bye
**
```

In UDC file APUDC, the following UDC's exist (among others):

```
Payable
Option NoBreak
File TODAT=TODAT.GLXEQ;Shr
File AP830WRK=VENNAMES;Rec=-80,16,F,ASCII;Disc=32,1,Save
Run PAYABLEX;Lib=G
**
VendorMaint
Option NoBreak
Display "Use this program with extreme care.<ctrl>G"
Run AP429X;Lib=G
**
```

Items of interest in the accounts payable examples:

1. There are two classes of users for most of the application systems: A "data entry" user who only needs access to one main menu-driven program (for example, PAYABLEX) and a user who needs access to that same program, as well as other utility programs (for example, AP429X) for the application system. I have implemented this by creating two UDC files. The "data entry" user has both UDC files enabled in the order listed above. However, the other user only has the second UDC file enabled. In this way, the UDC to actually run the program PAYABLEX only needs to be entered in the second UDC file (in this example that UDC is relatively short; however, some others have numerous FILE commands), with a reference made to it in the first, rather than needing to define it twice.

2. Even though the "data entry" user has both UDC files enabled for them, they effectively only have one thing they can do: RUN the PAYABLEX program. This is because of the LOGON execution option which automatically starts up the program at logon time, and logs them off (because of the BYE command) when the program stops running. Two more items are required to guarantee that this will work, however. The NOBREAK execution option must be specified to keep the user from pressing the BREAK key and gaining access to MPE. The CONTINUE command is also needed, so that if the PAYABLEX program should abort for some reason (I know - you write perfect programs that never abort - I have problems sometimes though!), the next command (in this case, BYE) will still execute.

Here is a neat idea for use with the SORT program (it works well with many others too!):

```
EquipmentSort
Purge EQPSORTD
Run SORT.PUB.SYS;StdIn=SDEQPSRT;StdList=$NULL
**
```

Most any program can have its standard input and output files redirected (i.e., provided from, or sent to, some place other than normal). See the RUN command for more details on this. In this UDC, the interactive input is coming from a disc file named SDEQPSRT, and the interactive output is discarded. File SDEQPSRT contains the commands that would normally have been provided to the SORT program (i.e., the names of the input and output files, and the key information).

Now, we will look at some UDC's from our job costing system. It has not been converted to an interactive system yet, and is still using the ENTRY.PUB.SYS program to create batch files which are then processed. The UDC's do provide some good examples of nesting UDC's, however.

```
EditJobList BFName
Option NoBreak
Reformat !BFName
Sort !BFName, SDEDTLST
File JCWRK,New;Temp
File CARD=SORTFILE,OldTemp
Run JC010X
File JCWRK,OldTemp
File JCMSTR,Old
File LIST=JC020LST;Cctl;Dev=LP
Run JC020X
**
EditWeeklyCost BFNAME
Option NoBreak
Reformat !BFName
Sort !BFName,SDEDTCST
File CARD=SORTFILE,OldTemp
Run JC110X
**
Reformat BFName
Purge !BFName,Temp
File REFFILE=JCREF.REFFILE,Old
File BATCH=!BFName.FCDATA,Old
File OUTFILE=!BFName,New;Temp
File REFLIST=$NULL
Run REFORMAT.PUB.SYS
**
```

```
SORT BFName,SDName
Purge SORTFILE,Temp
File INPUT=!BFName,OldTemp
File OUTPUT=SORTFILE,New;Temp
Run SORT.PUB.SYS;StdIn=!SDName;StdList=$NULL
**
```

Items of interest in these job costing UDC's:

1. UDC's EDITJOBLIST and EDITWEEKLYCOST both reference UDC's REFORMAT and SORT; however, different information is provided for the parameter values, so they process the files differently.

2. UDC REFORMAT only needs to know what the name of the batch file is that was created with ENTRY.PUB.SYS. The output from the REFORMAT program is stored in a temporary file with the original batch file name.

3. UDC SORT needs to know the batch file name and the name of the "sort data" file. The input data file for the sort is actually the temporary file created while reformatting. The output data file is always named SORTFILE, and is a temporary file. Temporary files are used for two reasons: first, they are automatically PURGEd when the session or job logs off; and secondly, they permit two sessions or jobs to do the same thing at the same time and eliminate the need to worry about permanent file usage conflicts. The "sort data" file contains information about how to sort the input file, and is specified by using the STDIN parameter of the RUN command (see the previous example for further discussion of this).

Here are some examples from the financial system that show how to execute the same program, but provide for different input and/or output.

```
BalanceSheet
BalShRun 11,";Forms=BLANK8 1/2 by 11 INCH PAPER REQ <ctrl>G."
**
BalanceSheetGB
BalShRun 2
**
BalShRun Copies,Forms=" "
File BALSHLST;Dev=LP,,!Copies;!Forms
Run BALSHX
**
CorporatePL
PrintPL C,12
**
DivisionPL
PrintPL 1,13
**
```

```
ClientPL
PrintPL 2,9
**
PrintPL FileCode,Copies=1
File PLFLA=PLFL!FileCode,Old
File PRPALLST=PRPL!"FileCode"LST;Dev=LP,9,!Copies;&
    Forms=BLANK 8 1/2 by 11 PAPER REQ <ctrl>G.
Run PRPALX
**
```

Here is what to look for in these examples:

1. There are seven UDC's, but only two actually RUN a program; the other five just reference those two.

2. In the case of the balance sheet program (BALSHX), we need to print the report on both blank paper and regular "green bar" paper. The program takes very little time to run, so we run it twice, once with each UDC, to create two different spool files with a different number of copies for each. In one case, the FORMS parameter and value is provided, and in the other case, the default value is used.

3. In the case of the profit and loss program (PRPALX), we need to print a different number of copies of each type of statement. Additionally, because of the way the program is set up, there is a different input file for each type of statement. There is actually only one character different in each file name, so the FILECODE parameter is used to provide that.

4. In UDC PRINTPL, notice the use of quotation marks (") around the FILECODE parameter. This is because it needs to be substituted in the middle of a "word".

5. In that same UDC, also notice the use of the ampersand character (&) to continue the logical line onto the next physical line of the UDC. Were all of that to have been typed on the same line, it would have gone beyond the seventy-two character limit for a line.

What if you have two different users that need to use the same program, but do not use the same "terminology"? Create two UDC's that RUN the same program. Our invoicing/accounts receivable system provides a good example. We have one set of users that work with the invoicing part of the system and a second set that works with the accounts receivable part of the system. It is really all one application system, but the two user groups tend to think of them as (related, but) separate entities. So. . .

```
Invoices
InvRecRun
**
ARec
InvRecRun
**
InvRecRun
Option NoBreak
File ENAME.PUB;Shr
File TODAT=TODAT.GLXEQ;Shr
Run INVRECX;Lib=G
**
```

In this example, each user group has a UDC that makes sense to them but both end up at exactly the same place!

Since most of you are probably programmers, let us take a look at some UDC's that you may find useful in your day-to-day existence. This list is not meant to be exhaustive, but rather meant to get you started:

```
DBLoad
Option List
Run DBLOAD.PUB.SYS
**
DBSchema SchemaFile
Option List
File DBSTEXT=!SchemaFile
File DBSLIST;Dev=LP
Run DBSCHEMA.PUB.SYS;Parm=3
Reset DBSTEXT
Reset DBSLIST
**
DBUnLoad
Option List
Run DBUNLOAD.PUB.SYS
**
DBUtil
Option List
Run DBUTIL.PUB.SYS
**
FormSpec
Option List
Run FORMSPEC.PUB.SYS
**
Preps ProgName,MaxDataSize=15000
Option List
Purge !"ProgName"X
Prep !"ProgName"U,"ProgName"X;MaxData=!MaxDataSize
Save !"ProgName"X
**
```

```
Query
Option List
Run QUERY.PUB.SYS
**
RunLG Program
Option List
Run !"Program"X;Lib=G
**
```

There is not a whole lot to explain here -- nothing fancy -- just some examples of commands and tasks that you probably do most every day. The only item I will mention is the use of OPTION LIST which displays the command as it is executed. This helps remind me that I am using a UDC and that some day, some where, I may need to type this longer command.

Now for some UDC's that the system operator would likely find useful:

```
AfterChecks
OptionList
DownLoad 6,VFCSTD6.PUB.SYS
DownLoad 6,Margin=1
StartSpool 6
**
BackUp
Option List
Limit 2,1
File BCKUPCNF.BACKUP=BCKUPCNF.PUB.SYS
Run BACKUP.HPUNSUP.SUPPORT;Info=" "
AbortJob HPTREND,MGR.TELESUP
Limit 1,1
**
BeforeChecks
Option List
Continue
StartSpool LP
StopSpool 6
HeadOff 6
**
JobF Priority=0
Option List
JobFence !Priority
**
LoadVFC VFCFileName=VFCSTD6
Option List
DownLoad 6,!VFCFileName.PUB.SYS
**
Margin LeftColumn=1
Option List
DownLoad 6,Margin=!LeftColumn
**
```

```
OffSites
Option List
Limit 2,1
Stream OFFSITEJ.PUB.SYS
AbortJob HPTREND,MGR.TELESUP
Limit 1,1
**
OutF Priority=1
Option List
OutFence !Priority
**
RCop FileNumber,Copies=1
Option List
AltSpoolFile #O!FileNumber;Copies=!Copies
**
RDef FileNumber
Option List
AltSpoolFile #O!FileNumber;Pri=0
**
RPri FileNumber,Priority=8
Option List
AltSpoolFile #O!FileNumber;Pri=!Priority
**
RPur FileNumber
Option List
DeleteSpoolFile #O!FileNumber
**
StopSp Device=6
Option List
StopSpool !Device;OpenQ
**
StrtSp Device=6
Option List
StartSpool !Device
**
SysDown
Option List
ShowJob
ShowOut Job=@;Sp
Console
ShowTime
**
SetOpKeys
SFK 1 0 " Reply  " "         " 6 "Reply "
SFK 2 0 "  Redo  " "         " 4 "Redo"
SFK 3 0 "  Show  " " Cache   " 9 "ShowCache"
SFK 4 0 " System " "   Up    " 5 "SysUp"
SFK 5 0 "  Show  " "Reports " 17 "ShowOut Job=@; Sp"
SFK 6 0 "  Run   " " Spook5 " 18 "Run Spook5.Pub.Sys"
SFK 7 0 " Spook5 " " Detail " 8 "S @.@;@0"
SFK 8 0 " System " "  Down  " 7 "SysDown"
UserKeys
**
```

Just the same as the last examples, this is by no means an exhaustive list, but rather a starting point for you as you consider what makes sense on your system. Now for some items of interest:

1. We print our checks "hot" to the printer. That is, we stop the spooler process for the printer itself. Two UDC's make life easier in this situation: BEFORECHECKS and AFTERCHECKS. Nothing fancy; just some steps that have to be done over and over -- and done CORRECTLY every time.

2. BACKUP and OFFSITES are two UDC's that assist with performing SYSDUMPs. The BACKUP program referenced is a program that used to exist in the HPUNSUP group of the SUPPORT account before the days of the TELESUP account. Its sole purpose in life is to create and stream a batch job that either performs a full or a partial backup based on the day of the week and the information stored in the file BCKUPCNF. In recent years, HP has provided us with the FULLBACKUP and PARTBACKUP commands which nearly eliminated my need for the BACKUP program. My problem is that we perform two full backups every week: one stays on site; the other goes off site. The PARTBACKUP command performs a partial SYSDUMP since the last full backup. However, since my second full backup is off site, I really want the partial backups done after it to still go back to the last on site full backup. The BACKUP program permits me to handle this, the HP commands do not. At any rate, each of the UDC's set the job and session limits low, stream an appropriate job stream, and then abort the HPTREND job stream so that it is not running during the backup.

3. In the SETOPKEYS UDC, the SFK UDC is referenced. This will be discussed later. For the time being, just understand that the SETOPKEYS UDC will cause the terminal function keys to be loaded with this information.

If you are getting tired of examples, feel free to go on to the next section. For those of you who want more examples, we will next look at some of the UDC's from my system level UDC file, and then finish up with some unique "goodies" that may prove useful to some of you:

```
AboJ JSNumber
Option List
AbortJob #!JSNumber
**
AltJ JobNumber,InPriority=8
Option List
AltJob #J!JobNumber;InPri=!InPriority
**
Con LDev=" "
Option List
Console !LDev
**
```

```
Ed
Option List
Editor
**
Entry
Option List,NoBreak
ListF BF@,0
Run ENTRY.PUB.SYS
**
Files
Option LogOn, List
File LP;Dev=LP
File T;Dev=TAPE
Display "The above File commands are in effect."
Display "The system UDC's are enabled."
**
J WHAT2SHOW=" "
Option List
ShowJob Job=@!WHAT2SHOW
**
L FileSet="@",Detail=2,ListFile=$STDLIST
Option List
ListF !FileSet,!Detail;!ListFile
**
LEq
Option List
ListEq
**
List FileName,ListFile=$STDLIST
Option List
FCopy From=!FileName;To=!ListFile
**
LT FileSet="@",Detail=2,ListFile=$STDLIST
Option List
ListFTemp !FileSet,!Detail;!ListFile
**
LUDC
Option List
ShowCatalog
**
Me
Option List
ShowMe
**
Out Items="Sp"
Option List
ShowOut Job=@;!Items
**
Print FileName,Copies=1,Priority=8
Option List
File LISTING;DEV=LP,!Priority,!Copies
FCopy From=!FileName;To=*LISTING
Reset LISTING
**
```

```
PScreen
Option List
Run PSCREEN.PUB.TELESUP
**
Purges FL1,FL2=$NULL,FL3=$NULL,FL4=$NULL,FL5=$NULL,FL6=$NULL
Option List
Purge !FL1
Purge !FL2
Purge !FL3
Purge !FL4
Purge !FL5
Purge !FL6
**
Res
Option List
Resume
**
RunPS Program
Option List
Run !Program.PUB.SYS
**
SetUDC File1=$NULL,File2=$NULL,File3=$NULL,File4=$NULL,File5=$NULL
Option List
SetCatalog !File1,!File2,!File3,!File4,!File5
**
ShC
Option List
ShowCache
**
ShD LDev=" "
Option List
ShowDev !LDev
**
Sort Input,Output
Option List
File INPUT=!Input
File OUTPUT=!Output
Reset LIST
Run SORT.PUB.SYS
**
Spook5
Option List
Run SPOOK5.PUB.SYS
**
Str JobName,Char="!"
Option List
Stream !JobName,!Char
**
StrAt JobName,Time,Char="!"
Option List
Stream !JobName,!Char;At=!Time
**
```

```
StrDay JobName,Day,Time="0:0",Char="!"
Option List
Stream !JobName,!Char;Day=!Day;At=!Time
**
StrIn JobName,Days=0,Hours=0,Minutes=0,Char="!"
Option List
Stream !JobName,!Char;In=!Days,!Hours,!Minutes
**
T
Option List
ShowTime
**
Display Message=" "
Option List
Comment <esc>M!Message
**
```

A few notes and points of interest:

1. In the few situations where we still use the ENTRY program to create batch files, we use the naming convention of always starting the batch file name with the letters "BF". In the ENTRY UDC, the purpose of the LISTF command is just to provide a list of existing batch files to the user before the program starts running.

2. When I want a quick listing of a file on my screen or on paper, I use the LIST or PRINT UDC's. Both use the FCOPY command to produce the listing. One slight inconvenience (besides the "extra garbage" displayed): unless the file record length is eighty (for LIST) or 132 (for PRINT) characters, FCOPY gives you a warning message that you must respond to.

3. The DISPLAY UDC is used to display the character string provided in the MESSAGE parameter. The escape M sequence causes the cursor to delete the line that the cursor is on, then display the message. The end result is that only the message is left on the screen. Note that this UDC is at the end of the UDC file so that all other UDC's may use it to display messages.

O.K. Now for those unique "goodies" I promised you. Have you been looking for a way to load information into your terminal's function keys automatically? If you have, keep reading. If you have not, perhaps you want to know why you would want to. Using the function keys to execute commands is a good alternative to UDC's because the overall overhead is usually less. Function keys also provide a one or two keystroke execution of commands. Why am I telling you about an alternative to UDC's?!? Because the solution is accomplished with UDC's.

My original source for this information was from the November 1987 issue of the Interact magazine. In that issue, Michael J. Parker and Lynn

Wilson of State Farm Insurance in Bloomington, Illinois, had a short article in the Users' Forum section of the magazine. I have taken their ideas and expanded them to work on all of the types of terminals I have (HP 2645A, HP 2392A, and HP 700/92), and, I believe, on any HP terminal. There are three parts to the solution: two UDC's that need to be defined once -- probably in a system level UDC, and one or more additional UDC's defined that use the first two. The two system level UDC's should be defined as:

```
SFK Key=1;Attr=0;Head1="        ";Head2="        ";&
    Length=40;Function="                                        "
Option List
Comment <esc>&f!"Attr"a!"Key"k!"Length"L!Function<esc>M<esc>A
Comment <esc>&f!"Key"k16d0L!Head1!Head2<esc>M<esc>A
**
UserKeys
Option List
Comment <esc>&jB<esc>M<esc>A
**
```

The SFK UDC accepts the information for one function key and "loads" that information by causing it to be displayed on the terminal with the COMMENT command. Be careful when you type this one in: upper and lower case makes a difference in how it will execute! The KEY parameter signifies which function key (1 through 8). The ATTR parameter indicates what should happen when the function key is pressed:

0 = (Normal) The defined string is displayed. To execute it, the user must press the RETURN key.
1 = (Local Only) The defined string is displayed; however, it may not be executed.
2 = (Transmit) The defined string is displayed and immediately executed.

The HEAD1 and HEAD2 parameters provide values to be placed in the labels on the screen (only for terminals that can "label" the function keys). LENGTH indicates how many characters are in the function string. And finally, the FUNCTION parameter provides the actual character string to be "loaded" into the function key.

I have used the COMMENT command twice in this UDC because we have a mixture of terminals and not all of them have the capability of labelling the function keys. The first (longer) COMMENT command will work on all of the terminals and will cause all of the information to be loaded except for the function key labels on the screen. The second COMMENT command provides the additional label information to those terminals that can accept it (the older model terminals just ignore it). It must be split in two steps; if combined into one, the older model terminals will not have any of the information loaded into the function keys. If all of your terminals have the function key labelling capability, you may combine them into one COMMENT. On the other hand, if none of your terminals have this capability, the second COMMENT could be left out and the HEAD1 and HEAD2 parameters could be eliminated.

The USERKEYS UDC simply causes the function key labels to be displayed on the terminal screen (if they are not already). Again, this is ignored by the older model terminals, but is needed for the newer ones. The "<esc>M<esc>A" sequence in both UDC's effectively "erases" the comments from the screen as the UDC executes. If you would like them left on the screen, that sequence could be left off the end of the line.

What might the UDC's that use these look like? Here is an example:

```
SetMainKeys
SFK 1,0,"        "," QUERY  ",,"Run QUERY.PUB.SYS"
SFK 2,0,"        "," DBUTIL ",,"Run DBUTIL.PUB.SYS"
SFK 3,2,"        ","SHOWJOB ",,"ShowJob"
SFK 4,0,"        ","FORMSPEC",,"Run FORMSPEC.PUB.SYS"
SKF 5,0,"        "," EDITOR ",,"EDITOR"
SFK 6,0,"        "," SPOOK  ",,"Run SPOOK5.PUB.SYS"
SFK 7,0,"        ","SEGMENTR",,"Run SEGMENTER.PUB.SYS"
SFK 8,0,"PrepSave","Program ",,"Preps "
UserKeys
**
```

If you want to take this a step further (although you do get back your saved UDC overhead), instead of RUNning each of the above programs, execute a UDC to do so. In the UDC for each program, execute a UDC before and after the RUN command to load the keys for the program about to be run and then reset them afterward. For example:

```
Query
SetQueryKeys
Run QUERY.PUB.SYS
SetMainKeys
**
DBUtil
SetDBUtilKeys
Run DBUTIL.PUB.SYS
SetMainKeys
**
```

Do remember to place these in the UDC file prior to the SETMAINKEYS and each of the SETxKEYS UDC's. By using this nesting technique, you can set up a "menu" system with only UDC's and terminal function keys. Neat, huh?!?

Now for "goodie" number two. Have you ever wanted to get rid of a set of files but did not really want to type PURGE over and over and did not have access to MPEX? Now you can (maybe)! This idea came from the March 1988 issue of The Chronicle newspaper. Victoria Shoemaker (of Taurus Software) in her :NEWUSER column, presented this novel solution to the problem:

```
PurgeFS FileSet
Option List
Store !FileSet;$Null;Show;Purge
**
```

This UDC STOREs the file set you specify to $NULL (which does nothing -- you do not even need to REPLY to a request!) and then PURGEs the files afterwards. This format of the STORE command is normally used to archive information and then remove it from the system -- we just happen to be archiving to the "bit bucket". Any file set that the STORE program will accept (including the "-" option) can be provided to this UDC. Be careful with this one -- it can be very powerful. Make sure you have a good backup before you type PURGEFS "@.@.@"!

"Goodie" number three. Do you need a way to provide different "welcome" and/or "news" messages for each user? If so, read on. This one comes from an article M.E. Kabay (of JINBU Corporation) wrote in the March 1988 issue of The Chronicle. Here's a system level UDC:

```
SysMessage
Option LogOn
Run LIST.PUB.TELESUP;INFO="W ON;T OFF;L NEWS.PUB.SYS"
**
```

If an account needs a special message or "news" file, set up an account level UDC for them:

```
AcctMessage
Option LogOn
Run LIST.PUB.TELESUP;INFO="W ON;T OFF;L NEWS.PUB.ACCT"
**
```

Special needs for a user? Try this user level UDC:

```
UserMessage
Option LogOn
Run LIST.PUB.TELESUP;INFO="W ON;T OFF;L NEWS.GROUP.ACCT"
**
```

The FCOPY program could be used instead of the LIST program, but I think you'll find the LIST program a little "nicer". By having all of these separate files, you can easily provide different information to different users or accounts by including it in their own news file. To update a file, simply use your favorite text editor.

One last "goodie". This one permits you to send a message or list the contents of an entire file on any terminal screen that is turned on but not logged on:

```
Send LDev,Source
File TERM;Dev=!LDev
Continue
FCopy From=!Source;To=*TERM
Reset TERM
**
```

This UDC, when executed, uses the FCOPY program to copy the specified file to the specified terminal. Again, the destination terminal must be turned on but logged off for the message to be displayed. If $STDIN is provided for the SOURCE parameter, then the user may type whatever he/she wishes at the time of the UDC execution. (This can be a little tricky though -- there is no prompt and you must type ":EOD" to end your message.)

A situation I find this UDC helpful in is after a backup has completed, and I need to let users know that they can logon. Consider the following:

```
SendToAll Source
Send 22,!Source
Send 23,!Source
Send 24,!Source
Send 25,!Source
Send 26,!Source
Send 27,!Source
Send 28,!Source
Send 29,!Source
Send 30,!Source
Send 31,!Source
Send 32,!Source
Send 33,!Source
Send 34,!Source
Send 35,!Source
**
```

If I type:

```
SENDTOALL SYSTEMUP.PUB.SYS
```

then it will attempt to transmit the contents of file SYSTEMUP.PUB.SYS to each terminal. If a particular terminal is not turned on, or is already logged on, that FCOPY will fail, but because of the CONTINUE command, the next one will still be attempted.

2.4 What are some problems I may have while using UDC's?

Throughout the paper, I have provided a number of warnings and "gotchas". Listed here (in somewhat random order) are a few worth repeating and a few not mentioned previously.

If a system and a user level UDC have identical names and an account level UDC references a UDC by this name, then the system level UDC will be executed because of the UDC hierarchy.

If an error or warning occurs as a UDC executes, MPE will:

1. Print an appropriate error message.

2. Unless NOHELP is specified, print a caret (^) pointing to the error.

3. Unless NOHELP and NOLIST are specified, print the line in which the error occurred.

Every time a user logs on, a UDC directory is created for that session or job. If an error occurs during this initialization, only the UDC level in which the error occurs will fail to be initialized. All others will still be enabled.

A UDC name may not be "RFA" or start with the letters "RFA". This is reserved for HP's internal use. Any UDC that is "RFA" or starts with "RFA" will not execute and will result in the error message:

UNKNOWN COMMAND NAME (CIERROR 975)

If the SETCATALOG is executed as part of a UDC, it will be the last command executed in the UDC body. Additionally, if the SETCATALOG was part of a nested UDC, all levels of UDC execution are terminated after completion of the SETCATALOG command.

If you execute a UDC that RUNs a program, you press BREAK, and then execute the SETCATALOG command while in BREAK, you may type RESUME, and continue with that program; however, any further execution of the UDC that issued the RUN command will be terminated.

UDC's are not always as secure as you might think. Certain programs and subsystems (e.g., SPOOK) allow users to enter MPE commands and RUN programs. So even if you have a UDC with LOGON and NOBREAK specified, the user can still gain access to MPE.

When you run a program, you no longer have access to your UDC's. The COMMAND intrinsic only can be used to execute MPE commands.

Consider using the CONTINUE command wherever possible. This will help prevent a program from aborting and terminating the UDC execution. Even though it may seem unlikely that a program will abort, it can be accomplished in many programs by typing :EOD when prompted for input. This causes an end of file condition on $STDIN and gives many programs problems.

Even though you have disallowed certain commands to a user (by redefining them with a UDC), be careful. If the user has access to

the COMMAND intrinsic (e.g., through EDITOR), the user can still execute most MPE commands.

When modifying a UDC file, make sure that all users (sessions and jobs) accessing that file have either logged off or disabled UDC's with the SETCATALOG command. If you are working on a system level UDC file, that means every user on the system is affected!

If you get rid of a user (with the PURGEUSER command) that had UDC's enabled at the time, the entries are NOT removed from COMMAND.PUB.SYS. Always execute a SETCATALOG command to disable UDC's for the user before purging the user.

UDC's bring with them system resource overhead at logon time, and they use up entries in the DST table. The DST table entries are used because of the extra data segment used to store the UDC directory for each job or session. To reduce overhead and improve system performance when UDC's are used, do whatever you can to reduce the number of UDC files. This will reduce the number of DST's used, as well as reduce the number of FOPEN's.

Here is a question you have not asked yet: is there a maximum number of UDC's that may be enabled for a user? The answer is yes. Every UDC enabled for a job or session must have an entry placed in the UDC directory in an extra data segment. When that extra data segment becomes full, that is the maximum number of UDC's.

If you keep these potential problems and limitations in mind as you start your adventure into the wonderful world of UDC's, you should do well in avoiding most of the problems and pitfalls that may arise along the way.

2.5 Are we done with UDC's yet!?!

Yes! At least for the time being. And now for something completely different. . .

3.0 What are Job Control Words (JCW's)?

JCW is one of the many other acronyms used in the HP 3000 world. This one comes from the phrase Job Control Word. A JCW is MPE's way of permitting programs and commands to communicate with each other within a given job or session. JCW's are unsigned integer variables used at the operating system level with values ranging from zero through 65,535. Each JCW has a name and can be set and/or interrogated either by MPE commands and/or programs.

3.1 How can JCW's help me?
-or-
Why would I want a program to talk to my commands?

Good questions! A properly used JCW will permit you to create "smarter" job streams. They can help to automate some of the decision making process in procedures. They can even help you catch errors before they become a problem! All of this is to say: JCW's can help make the system more "user friendly".

By testing JCW's against specific values, the user can program conditional statements that take action(s) based on the results of the test. JCW's can be set to predetermined values to indicate completion of steps within a procedure. JCW's can be checked to determine if certain events (usually errors) have occurred within MPE.

3.2 O.K. I think I see how they could help me. So, how do I create and use a JCW?

First, some background information. Three classes of JCW's exist: user-defined JCW's; system-defined JCW's; and system-reserved JCW's. In some ways, they are exactly the same -- in other ways, they are completely different.

User-defined JCW's are named and assigned values solely by the user. MPE never changes the value of, or interrogates a user-defined JCW. The user creates and assigns a value to this class of JCW's with the SETJCW command or the PUTJCW intrinsic. The JCW name must begin with an alphabetic character and may consist of a maximum of 255 alphabetic or numeric characters. You may not begin a JCW name with the mnemonic names OK, WARN, FATAL, or SYSTEM except under very specific conditions. (If you want to know what they are, see the HP "commands" manual.) The value assigned to a user-defined JCW must be in the range of zero to 65,535 inclusive. User-defined JCW's may be interrogated by the user with the SHOWJCW command and the FINDJCW intrinsic. These new commands and intrinsics will be discussed later, so please be patient.

System defined JCW's are named by the system and assigned values by the system and/or by the user. Both the system and the user may interrogate system-defined JCW's. Only two system-defined JCW's exist: JCW and CIERROR. Both are created and set to zero at the beginning of every job or session. They will remain zero unless the user changes their value or an error occurs. The JCW named JCW has two special values:

%140000 (System 0)	Program aborted per user request.
a value greater than %140000	Program terminated in an error state.

The CIERROR JCW keeps track of the command interpreter (CI) errors. If a CI error occurs, CIERROR is set to reflect the most recent error number. Valid commands do not reset CIERROR to zero. Thus, it always contains the number of the last error that occurred, unless the user resets its value. Generally, it is best not to alter the values of the system-defined JCW's. If you need to control a JCW, it is best to use a user-defined JCW.

System-reserved JCW's are named and assigned values solely by the system. Users may not change the value of a system-reserved JCW. They may, however, interrogate it. There are six system-reserved JCW's: HPMINUTE, HPHOUR, HPDAY, HPDATE, HPMONTH, and HPYEAR. The following briefly explains what each is:

HPMINUTE	Minute of the hour: values are 0 through 59.
HPHOUR	Hour of the day: values are 0 through 23.
HPDAY	Day of the week: values are 1 through 7; Sunday = 1.
HPDATE	Day of the month: values are 1 through 31.
HPMONTH	Month of the year: values are 1 through 12; January = 1.
HPYEAR	Year of the century: values are 0 through 99.

3.21 JCW usage in jobs and/or sessions.

O.K. So now you know what JCW's are. You even know about the three classes of JCW's and what who can do to what. But, how do you look at or set their values? For jobs or sessions, the answer is: with the SHOWJCW and SETJCW commands.

The SHOWJCW command displays the current value of one or more JCW's. Its syntax is:

```
SHOWJCW [jcwname]
```

where "jcwname" is a valid JCW name (any class). If a name is provided, then only the value for that JCW will be displayed. If a name is not provided, then all system-defined and user-defined JCW's and their values are displayed -- system-reserved JCW's are not displayed. This command may be executed from a session, job, in BREAK, or from a program. It is BREAKable (it aborts execution of the command).

If no user-defined JCW's have been created and the user types:

```
SHOWJCW
```

the system will respond with

```
JCW=0
CIERROR=0
```

unless some error has occurred prior to this command.

If you wish to see the current value of a specific JCW, you might type:

```
SHOWJCW HPDAY
```

and the system would respond with:

```
HPDAY=3  SYSTEM RESERVED JCW
```

Or, if you typed:

```
SHOWJCW UPDATEERRORS
```

and UPDATEERRORS was a valid user-defined JCW name, the system would respond with:

```
UPDATEERRORS=2
```

Big deal, so you can look at the value of a JCW. So what?!? O.K., let me tell you how you can change or set the value of a JCW -- that is a little more productive. We need the SETJCW command to do this:

```
                          [+value]
SETJCW jcwname=value      [-value]
```

where "jcwname" is the name of a new or existing user- or system-defined JCW and "value" represents one of the following:

1. An octal number between zero and %177777, inclusive.
2. A decimal number between zero and 65,535, inclusive.
3. An MPE-defined JCW value mnemonic (OK, WARN, FATAL, or SYSTEM)
4. The name of an existing JCW.

All values must be in the range of 0 to 65,535, inclusive. That is, if the "+" or "-" option is used, the result of the arithmetic must be in the range as well. (The equal sign following the "jcwname" may actually be one or more punctuation characters or spaces, except "%" and "-". If you prefer some other notation, feel free. . .) This command may be executed from a session, job, in BREAK, or from a program. It is not BREAKable.

A word or two about the four JCW value mnemonics. They are:

```
OK      value is zero
WARN    value is 16,384
FATAL   value is 32,768
SYSTEM  value is 49,152
```

These are strictly mnemonics for specific values -- they cannot be used as JCW names. You may use a combination of a mnemonic and a number to indicate a value between two mnemonics. If you specify:

```
FATAL32
```

for example, an implied addition takes place (32,768 + 32) and the value would be 32,800. The "+" and "-" option may also be used with mnemonics. For example:

```
FATAL - 768
```

would result in a value of 32,000. If the SHOWJCW command is used to display current JCW values, and a value is greater than one of the mnemonics, then the value will be displayed as the mnemonic plus the amount over. For example, a value of 16,386 will be displayed as:

```
WARN2
```

An exception to this is that any value less than 16,384 will be shown as the actual number.

When the SETJCW command is executed, it causes the MPE JCW table to be scanned for the name of the specified JCW. If the name is found, the JCW is set to the value provided. If the name is not found, it is added to the JCW table and then set to the value provided. Once a JCW is created, it exists for the duration of that session or job. There is no way to delete a JCW, short of logging off.

You still are not feeling very productive with JCW's yet, right!?! O.K. Here is the good stuff. JCW's are most often used to control the flow of batch jobs (they can also be used in UDC's and/or in sessions), taking various actions based on the results of previous steps. To do this, the IF/THEN, ELSE, and ENDIF commands are used. For purposes of this paper, I am going to assume you either know how these MPE commands work or can easily acquire the knowledge as we look at examples. Some examples will come later that should clear up some of your questions.

3.22 JCW usage in programs.

Now, for you programmer-type people, we will take a look at how to interrogate and set JCW's from within a program. My examples will be based on COBOLII usage; however, I will try to provide information in a general way.

The programmatic equivalent to the SHOWJCW command is the FINDJCW intrinsic. Its syntax (in COBOLII format) is:

```
CALL INTRINSIC "FINDJCW" USING jcwname,jcwvalue,status
```

where "jcwname" is an alphanumeric variable (byte array) containing the name of the JCW to be found, "jcwvalue" is an unsigned one-word integer variable (logical) to which the JCW value is returned and "status" is a signed one-word integer variable (integer) to which a value denoting the execution status of the intrinsic is returned. The "jcwname" parameter may contain up to 255 alphanumeric characters, starting with a letter and ending with a nonalphanumeric character, such as a blank. If the requested JCW is found, its value is returned to the program in the "jcwvalue" parameter; if not found, no change is made to this parameter. The "status" parameter will be returned with one of four possible values:

0 Successful execution; the JCW was found.
1 Error: "jcwname" is longer than 255 characters.
2 Error: The value of "jcwname" does not start with a letter.
3 Error: The JCW was not found in the JCW table.

The FINDJCW intrinsic may be used to return the value of any of the three classes of JCW's.

The SETJCW command's programmatic equivalent is the PUTJCW intrinsic. Its syntax (in COBOLII format) is:

```
CALL INTRINSIC "PUTJCW" USING jcwname,jcwvalue,status
```

where "jcwname" is an alphanumeric variable (byte array) containing the name of the JCW to be created or changed, "jcwvalue" is an unsigned one-word integer variable (logical) containing the value for the JCW, and "status" is a signed one-word integer variable (integer) to which a value denoting the execution status of the intrinsic is returned. The "jcwname" parameter may contain up to 255 alphanumeric characters, starting with a letter and ending with a nonalphanumeric character, such as a blank. If "@" is the value used, all JCW's will be set to the value provided. If the specified JCW already exists in the JCW table, its value is changed to the value provided in the "jcwvalue" parameter; if not there, an entry is created and then it is assigned the specified value. The "status" parameter will be returned with one of six possible values:

0 Successful execution; value entered in the JCW table.
1 Error: "jcwname" is longer than 255 characters.
2 Error: The value of "jcwname" does not start with a letter.
3 Error: JCW table overflow; no room to create this new JCW.
4 Error: Attempted to assign a value to a JCW value mnemonic.
5 Error: Attempted to assign a value to a system-reserved JCW.

The PUTJCW intrinsic may only be used to assign a value to a user-defined or system-defined JCW. As mentioned earlier, if the JCW named JCW is set to exactly %140000, then when the program stops running, the system will display:

```
PROGRAM ABORTED PER USER REQUEST (CIERR 989)
```

If it is set to any value greater than %140000, then the message displayed will be:

```
PROGRAM TERMINATED IN AN ERROR STATE (CIERR 976)
```

If the program is running in a batch job, the job will terminate unless a CONTINUE command precedes the RUN command.

Another possibility with JCW's is to use them to permit separate processes within the same job or session to communicate with each

other. If a process were to set a JCW to a given value when a certain event occurred, then any other related process could check that JCW to find out when it occurred or what has occurred. Remember, however, that only numeric information may be assigned to JCW's.

Just so you can't say that I didn't tell you about them, two other intrinsics exist: GETJCW and SETJCW. They only permit you to interrogate and set, respectively, the value of the system-defined JCW named JCW. Since they have limited usability and you can use FINDJCW and PUTJCW to accomplish the same thing, I do not suggest learning about them.

3.3 How about some examples?

Once again, you have a good idea! First, we will look at some examples in batch jobs, then we will take a look at a couple example programs that use JCW's.

3.31 Batch Job Examples.

Since most of us are likely programmers, I will start with an example job stream I use when working with a program. Virtually all programs that are created at N.G. Gilbert Corporation are written in the PROTOS language. For those of you not familiar with PROTOS, it is a program generator whose output is a complete, structured, COBOLII program. Once the program has been written in PROTOS (and keyed into an editor file), the next step is to have PROTOS create the COBOLII source program. When PROTOS completes that task, then you need to compile the COBOLII source program. The final step is to prepare the object program to produce the program file.

Since I do not want to tie up a terminal (and the system) while all of this transpires, I use an "intelligent" job stream to handle the various tasks.

```
!Job prog,PRGRMR.NGG
!Comment This job stream performs a ProWrite, COBOLII compile,
!Comment PREPare, and SAVE of a program.  If any errors
!Comment are encountered along the way, a message is sent
!Comment to PRGRMR.NGG, and the job stream stops running.
!SetJCW CIError = OK
!Continue
!ProWrite progP,prog,$STDLIST,2000
!If CIERROR < > OK Then
!   Tell PRGRMR.NGG; ProWrite of prog aborted<ctrl>G.
!Else
!   If PROTOSError < > OK Then
!      Tell PRGRMR.NGG; Errors in ProWrite of prog<ctrl>G.
```

```
!   Else
!      Tell PRGRMR.NGG; ProWrite of prog done. Compile started.
!      File COPYLIB=PROCOPY
!      Continue
!      COBOLII prog, progU
!      If CIError < > OK Then
!         Tell PRGRMR.NGG;Complie of prog aborted <ctrl>G.
!      Else
!         Tell PRGRMR.NGG; Compile of prog done. Prepare started.
!         Purge progX
!         Continue
!         Prep progU,progX;MaxData=20000
!         If CIError < > OK Then
!            Tell PRGRMR.NGG;Prepare of prog aborted <ctrl>G.
!         Else
!            Save progX
!            Tell PRGRMR.NGG;Program prog is done <ctrl>G.
!         EndIf
!      EndIf
!   EndIf
!EndIf
!EOJ
```

This example demonstrates how you can structure a job stream to check for errors and take different actions based on the occurrence or non-occurrence of errors. Some points of interest:

1. JCW CIERROR is set to OK (i.e., zero) at the start of the job to guarantee that it starts at zero.

2. Even if one of the programs (PROWRITE, COBOLII, or PREP (SEGMENTER) aborts, the job stream will continue because of the CONTINUE commands. This permits us to check CIERROR after each one to see if it aborted. If the CONTINUE commands were not used, the job stream would abort before we could check the JCW's.

3. PROTOSERROR is a JCW that the PROWRITE program creates and sets equal to the number of errors found in your PROTOS program. By checking it, we can determine whether it is worthwhile to continue on with the compile and prepare steps.

4. Notice that you may nest the IF statements to create whatever logic that might be required.

Here is another variation of this same job stream. This version just tells the programmer that there is an error, but does not report in detail like the previous version:

```
!Job prog,PRGRMR.NGG
!SetJCW CIError=OK
!SetJCW False=1
!SetJCW NoProblems=OK
!Continue
!ProWrite progP,prog,$STDLIST,2000
!If CIError < > OK Then
!   SetJCW NoProblems=False
!Else
!   If PROTOSError < > OK Then
!      SetJCW NoProblems=False
!   Else
!      File COPYLIB=PROCOPY
!      Continue
!      COBOLII prog, progU
!      If CIError < > OK Then
!         SetJCW NoProblems=False
!      Else
!         Purge progX
!         Continue
!         Prep progU,progX;MaxData=20000
!         If CIError < > OK Then
!            SetJCW NoProblems=False
!         Else
!            Save progX
!         EndIf
!      EndIf
!   EndIf
!EndIf
!If NoProblems=False
!   Tell PRGRMR.NGG;Errors in program prog <ctrl>G.
!Else
!   Tell PRGRMR.NGG;Program prog is done.
!EndIF
!EOJ
```

In this version, JCW NOPROBLEMS is set to indicate when an error has occurred and then is checked at the end of the job stream to determine what to tell the programmer. (Note the creation of JCW FALSE.)

Here is an example of making use of the system-reserved JCW's. This is a modified version of the job stream we use to do our daily backups.

```
!Job BackUp,Operator,Sys
!SetJCW Monday=2
!ShowAllocate
!Run FREE5.PUB.SYS
!ShowCache
!Report
!If HPDay=Monday Then
!   FullBackUp
```

```
!Else
!   PartBackUp
!EndIf
!Stream JHPTREND.HP35136A.TELESUP
!If HPDay=Monday or HPDay=Monday+3 Then
!   Stream PREDICTJ.HP05093A.TELESUP
!EndIf
!EOJ
```

Points of interest in the backup job stream:

1. Notice the creation of the JCW MONDAY. This is not necessary, but it makes the IF command read a little nicer.

2. If the JCW HPDAY has a value of 2, then it is Monday and a full backup should be performed. Otherwise, a partial is done.

3. When the backup has completed, the HP trend job stream is restarted, and if it is Monday or Thursday (Monday + 3), then the HP predict job stream is also started.

Here is an example of using the IF command and JCW's to do something they were not designed for, but it works, so why not?!! The following could be included at any point in a job stream where JCW would not be equal to FATAL (which would be virtually anywhere):

```
!If JCW=Fatal Then
     From this point on (up to an ELSE or ENDIF command), you
     may type whatever you wish.  The lines do not even need to
     start with an exclamation point!  The reason this works is
     that when the condition in an IF command is false (which
     it is in this instance), all command lines are ignored
     until an ELSE or ENDIF command is read.  Thus, this
     provides an easy way to include comments without using
     the COMMENT command.
!EndIf
```

Two other sources of example job streams are the PREDICTJ and JREDUCE job streams from HP. They make extensive use of JCW's to control the flow of the job stream logic.

3.32 Programmatic Examples.

The following is a COBOLII subroutine (actually it was written in PROTOS) I created to call in any situation where I want to end a program and have:

```
PROGRAM ABORTED PER USER REQUEST (CIERR 989)
```

displayed afterwards (granted, this is not very often!). The sole purpose of this program is to set JCW to a value of %140000.

```
$CONTROL DYNAMIC,BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SETABORTJCW.
 AUTHOR. DAVID L LARGENT.
 DATE-WRITTEN. TUE, OCT 20, 1987,   3:41 P.M.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

*****************************
*Sets JCW to "Program Abort".*
*****************************

 01 JCW-NAME                                  PIC X(4)
    VALUE "JCW " .
 01 JCW-STATUS                                PIC S9(4) COMP .
 01 JCW-VALUE                                 PIC 9(4) COMP
    VALUE %140000 .

 PROCEDURE DIVISION.

 MAIN-LINE-SECTION SECTION.

 MAIN-LINE.

     MOVE ZERO                          TO JCW-STATUS.
     CALL INTRINSIC "PUTJCW" USING JCW-NAME JCW-VALUE
          JCW-STATUS.
     IF JCW-STATUS NOT = ZERO
          DISPLAY "Program SETABORTJCW:   JCW not set."
     .
     GOBACK.
```

This may not be a very useful program to you, as it is printed; however, it does show the basics of what needs to be done to create and/or set a particular JCW to a given value. If your JCW name is longer than three characters, make sure you increase the length of field JCW-NAME. Also, make sure you have at least one blank or some other non-alphanumeric character following your JCW name. The value for JCW-VALUE can be specified as a decimal number if you prefer.

Here is a trivial example of the FINDJCW intrinsic, but again it shows the basics of what needs to be done to retrieve the value of an existing JCW.

```
$CONTROL BOUNDS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DISPLAYJCW.
 AUTHOR. DAVID L LARGENT.
```

```
       DATE-WRITTEN. WED, APR 27, 1988,  4:24 A.M.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      ****************************
      * Displays the Value of JCW *
      ****************************
        01 JCW-NAME                              PIC X(4)
           VALUE "JCW ".
        01 JCW-STATUS                            PIC S9(4) COMP.
        01 JCW-VALUE                             PIC 9(4) COMP.
        PROCEDURE DIVISION.

        MAIN-LINE-SECTION SECTION.

        MAIN-LINE.

           MOVE ZEROS                           TO JCW-STATUS
                                                   JCW-VALUE.

           CALL INTRINSIC "FINDJCW" USING JCW-NAME JCW-VALUE
                JCW-STATUS.
           IF JCW-STATUS NOT = ZERO
                DISPLAY "Program DISPLAYJCW: JCW not found."
           ELSE
                DISPLAY "JCW = " JCW-VALUE
           .
           STOP RUN.
```

Same comments as last time: make sure you set up JCW-NAME large enough to hold your JCW name and make sure you end it with at least one blank or other non-alphanumeric character.

3.4 What are some problems I may have while using JCW's?

Throughout the paper, I have provided a number of warnings and limitations. Listed here (in somewhat random order) are a few worth repeating.

A JCW name must start with a letter and may consist of a maximum of 255 alphabetic or numeric characters.

The value assigned (whether a number or a calculated value) must be in the range of zero to 65,535, inclusive.

The system-reserved JCW's may not be assigned a value by a program or the user.

If you want to check for program errors (e.g., program aborts), make sure to include a CONTINUE command before the RUN command.

Remember that the only thing that changes the values of CIERROR and JCW (except the user) is another error. That is, a valid command does not set them to zero!

Although JCW's can have any value from zero through 65,535, only values from zero through 16,383 will be displayed as numbers. Values larger than 16,383 will be displayed as a JCW value mnemonic and an offset beyond the mnemonic value.

4.0 How can I use UDC's and JCW's together?

UDC's and JCW's can be used together very effectively. JCW's can control the flow of logic within a UDC. A UDC (by way of its parameters) can gather information and use that information to set JCW's such that a program can then interrogate them and take appropriate actions. So, let us look at some examples of combining these two powerful capabilities.

To start off with, we will look at some logon UDC's. First, a simple addition to the system level logon UDC to make use of system-reserved JCW HPDAY easier:

```
SysLogOn
Option LogOn,NoBreak
SetJCW Sunday=1
SetJCW Monday=2
SetJCW Tuesday=3
SetJCW Wednesday=4
SetJCW Thursday=5
SetJCW Friday=6
SetJCW Saturday=7
**
```

By including this, these seven JCW's would always be available for use in IF commands. For example:

```
If HPDay=Monday Then
     FullBackUp
Else
     PartBackUp
EndIf
```

is easier to read and understood than "If HPDay=2...".

Next, a way to have one thing automatically happen if a session logs on and something else if a job logs on:

```
UserLogOn
Option Logon,NoBreak
SetJCW CIError=0
Continue
Resume
```

```
If CIError = 978 Then
     Run (batch program)
Else
     Run (online program)
     Bye
EndIf
**
```

The RESUME command is not allowed in job mode (thus, CI error 978), so we can use the result of its execution to determine if a session or a job is logging on. Also, note that a job could be set up to do other things after the "logon" program runs because there is not an automatic BYE like there is for the session.

Here is a way to control when people logon and play games:

```
GamesLogOn
Option LogOn,NoBreak
If HPDay = Sunday or HPDay = Saturday or &
     HPHour < 8 or HPHour > 17 or HPHour = 12 Then
     Display "Welcome to the Game Room"
Else
     Display "Sorry, the Game Room is closed."
     Display "Hours: Saturday and Sunday: all day"
     Display "       Monday-Friday: Before 8AM, After 5PM"
     Display "                      and Noon to 1PM"
     Bye
EndIf
**
```

If it is Saturday or Sunday, or before 8AM, after 5PM, or sometime during the noon hour, this UDC will let the user stay logged on. Any other time and the user will automatically be logged off. Note that if the user gets logged on during an "open" time, they may continue playing "forever" -- there is nothing to force them off when the game room "closes".

The following UDC is executed every morning by our operator or anytime the machine is restarted. It provides a quick, easy way of getting everything set to a known value:

```
SysUp
Option List
Streams 10
JobFence 0
JobPri CS,DS
JobSecurity Low
Continue
DiscRPS 3,Disable
Continue
StartCache 1
Continue
```

```
StartCache 2
SetJCW CIError=0
Continue
StartCache 3
If CIError=0 Then
     Stream JHPTREND.HP35136A.TELESUP
Else
     Display "<esc>&dJ The HPtrend job stream should be"
     Display "<esc>&dJ running.  If it is not,          "
     Display "<esc>&dJ STREAM JHPTREND.HP35136A.TELESUP"
EndIf
ShowJob JOB=@J
OutFence 1
HeadOff 6
Continue
StartSpool 6
Limit 3,16
SetOpKeys
**
```

A few points of interest:

1. Since we leave disc caching turned on virtually all of the time, this UDC assumes that if the "StartCache 3" command succeeds, the system must have just been started and, therefore, the trend job stream needs to be initiated.

2. The last thing that is done is to load the function keys on the system console with the SETOPKEYS UDC that was discussed earlier.

Here are a couple of UDC's to make things more convenient:

```
FindRun Program,Parm=0
SetJCW CIError=0
Continue
Run !Program.Group1;Parm=!Parm
If CIError=622 Then
     SetJCW CIError=0
     Continue
     Run !Program.Group2;Parm=!Parm
     If CIError=622 Then
          SetJCW CIError=0
          Continue
          Run !Program.Group3;Parm=!Parm
          If CIError=622 Then
               Display "This program was not found in"
               Display "group1, group2 or group 3."
          EndIf
     EndIf
EndIf
**
```

```
List FromFile=$StdIn,ToFile=$StdList
SetJCW CIError=0
Continue
File PRINT;Dev=!ToFile
If CIError=301 or CIError=344 Then
     File PRINT=!ToFile
EndIf
FCopy From=!FromFile;To=*Print
**
```

The FINDRUN UDC will try RUNning your program from three different groups before it "gives up". This could be changed to be different accounts as well, and could have more nesting added to try in more locations if you wish.

The LIST UDC provides a lot of flexibility. The "to" file can be specified by providing a device class name, a logical device number, or a file name! To list a file on your terminal, you would type:

```
LIST file
```

To print a file on the line printer, you could type either one of these:

```
LIST file,LP
LIST file,6
```

To list a file on a terminal with a logical device number of 27 (the terminal would need to be turned on, but logged off), you would type:

```
LIST file,27
```

To copy a file to an existing disc file, you would type:

```
LIST file,file2
```

It is not possible to create a new disc file. Another option available with this UDC is to use the default "FromFile" value of $STDIN, which will accept input from your terminal keyboard, thus permitting you to "create" a file as you go.

Now let us take a look at a UDC in which information is passed to a program by way of JCW's. This example is from the UDC's that PROTOS Software Company provides with their PROTOS system:

```
PROWRITE F,C="C",L=$NULL,S=1023,R=ROOTDB,Q=0
SETJCW PROBUILDWRITE=2
SETJCW QEDITOUT=!Q
FILE SEMDOPE=SEMDOP01.PROTOS.PROTOS;SHR
FILE SEMPASS=SEMPAS01.PROTOS.PROTOS;SHR
FILE SSERR=SSERR.PROTOS.PROTOS;SHR
```

```
FILE SEMTEMP=SEMTMP01.PROTOS.PROTOS;SHR
FILE ATNIN1=!F
FILE ATNOUT2 = !L
PURGE !C
BUILD !C;REC=-80,16,F,ASCII;CODE=EDTCT;DISC=!S
FILE COBOLOUT=!C
FILE ROOTDB=!R
IF QEDITOUT=1 THEN
     PURGE QECOBOUT,TEMP
     FILE QECOBOUT;REC=256;DISC=!S;TEMP
ENDIF
RUN PROTOS.PROTOS.PROTOS;LIB=G
IF QEDITOUT = 1 THEN
     PURGE !C
     RENAME QECOBOUT,!C,TEMP
     SAVE !C
ENDIF
RESET SEMDOPE
RESET SEMPASS
RESET SSERR
RESET SEMTEMP
RESET ATNIN1
RESET ATNOUT2
RESET COBOLOUT
RESET ROOTDB
**
```

In this example, there are two JCW's used to pass information to the PROTOS program: PROBUILDWRITE and QEDITOUT. The QEDITOUT JCW gets its value from the Q parameter of the UDC. Based on the QEDITOUT JCW, different parts of the UDC are executed.

JCW's may only be assigned numeric values. So how do you make use of character string information from a UDC parameter? Here is one way:

```
CopyW2ToTape CharSet=X
SetJCW A=0
SetJCW E=0
SetJCW !CharSet=1
If A=0 and E=0 Then
  Display "An 'A' or an 'E' must follow the copy command.<ctrl>G"
Else
  File W2TAPE;Dev=TAPE;Rec=-276,25,F,ASCII
  If A=1 Then
    FCopy From=PR997TAP;To=*W2TAPE
  Else
    FCopy From=PR997TAP;To=*W2TAPE;EBCDICOUT
  EndIf
  ListF PR997TAP,1
  Display "<esc>dJ Check the number of records copied.<ctrl>G"
EndIf
**
```

The purpose of this UDC is to copy a disc file to magnetic tape. The catch is that we need the option of copying it in EBCDIC format; thus, the CHARSET parameter. There are two "correct" responses: A and E. At the beginning of the UDC, two JCW's (A and E) are created and set to zero, then the user's choice is set to one. By having a default value that is not correct, if the user does not provide a value, the UDC will provide the message to the user and "remind" them of the correct values.

5.0 Closing Thoughts -- Is it worth the effort of learning something new?

We have looked at User Defined Commands and Job Control Words. Numerous examples have been explored to see how they work and how they can be used. They are a very powerful feature of the HP 3000. There is overhead associated with them (especially UDC's), yet, my feeling is that the convenience and "user-friendliness" gained outweighs that overhead. They must be controlled however -- people can get carried away when creating UDC's.

So, is it worth the effort? My answer is a (qualified) resounding YES! The qualification is that UDC's and JCW's must be carefully planned and monitored to reap the greatest benefit, but, oh, what a benefit it is: increased operator, programmer, and user productivity and a computer system that is easier to use overall.

Acknowledgements

A special thanks go to the following people who assisted in creating this paper in some (tremendous) way -- all of their efforts (and tolerance!) are greatly appreciated:

Dan Hinds
Jonathan Largent
Lois Largent
Karen Morgan
Dawn Thomas
Martha Walsh

Bibliography

Fisher, Sharon	"Setting up UCDs", Interact, July, 1984, page 48ff.
Griffin, Brad	"Another Way to SetJCWs as Part of a Log-on UDC", SuperGroup Association Magazine, June, 1986, page 12.
Hewlett Packard Company	Hewlett Packard Response Center Questions & Answers, August 1, 1986, page 1.
Hewlett Packard Company	MPE V Commands Reference Manual, First Edition, Update 1, 1986, Chapters 2 and 3.
Hewlett Packard Company	MPE V Intrinsics Reference Manual, Second Edition, Update 1, 1986, Chapters 2 and 5.
Hewlett Packard Company	System Operation and Resource Management Reference Manual, Second Edition, Update 1, 1986, Chapter 6.
Hewlett Packard Company Response Center (specifically B.A.)	Calls to the Atlanta HP Response Center during April, 1988.
Kabay, M.E.	"Making the Most of Your WELCOME Message", The Chronicle, March, 1988, page 54.
Largent, David L.	"Function Key Labelling", Interact, February, 1988, page 15.
Lund, Robert A.	"UDCs: A Primer", Interact, April, 1987, page 71ff.
Parker, Michael J. and Wilson, Lynn	"Labelling F-keys", Interact, November, 1987, page 22ff.
PROTOS Software Company	PROTOS Documentation, version 861231.
Shoemaker, Victoria	"UDCs: Marvelous and Misunderstood", The Chronicle, March, 1988, page 34.
Volokh, Eugene	"Burn Before Reading - HP3000 Security and You", Thoughts and Discourses on HP3000 Software, Third Edition, 1987, page 19ff.
Volokh, Eugene	"Conditional Execution -:IF, :ELSE, :ENDIF, ET. AL.", SuperGroup Association Magazine, February, 1986, page 28ff.