<OFFICE-4, TUTOR, TUTOR4.AUG;41,>, 24-Aug-81 07:17 DRH ;;;;

Introduction 1

This file is arranged in much the same way as was TUTOR3, but it tends to contain information that is not necessarily useful for everyone. As before, set your viewspecs to "x" to get an idea of the contents of this file. The material presented here is even more terse and compact than it was in TUTOR3, so please be prepared to reread certain sections over and over if they are not clear. 1A

Above all, do not start reading this file with the idea that you can finish it in one session and understand it thoroughly. Even after you have been using AUGMENT for quite some time, you may find it useful to come back to this document and reread sections. 1B

One final note before we start: Throughout this file, references are made to the "Executive". Depending on the type of machine you are working on, the name of the Executive will be either TENEX or AUGUST. For the purposes of this discussion, the differences between TENEX and AUGUST are not important, and you may read whichever one corresponds to your own Executive. 1C

File control 2

Naming files 2A

Before proceeding, it might be a good idea to review the three paragraphs beginning with <TUTOR3, 0227>. These discuss the various parts of a file name. 2A1

Think carefully before you assign a name to a new file. This may seem silly, but in fact, it is one of the most important things you can do to keep your files in order. The name you give a file should tell as much about the file as possible, and should remind you of its contents even if you have not looked at them for months. After a few months, it is not so easy to remember what the differences are between TOM1, TOM2, ..., TOM20. 2A2

It is also a good idea to try to make the names differ in the first few characters, rather than in the last ones. This is because the <ESC> key may be used for file name recognition as soon as enough text is typed to completely distinguish a name from any others. See also the section on <ESC>: <0243>. 2A3

If you have a number of similar files, in which the only difference in content is a date, or a customer name, or whatever, be sure to include that date or name in the file name. Examples might be: SEP79-ACCTS, AUG80-ACCTS, BLUM-INVENTORY, MILLER-INVENTORY, and so on. 2A4

In spite of your best intentions, you will occasionally find that you gave a file the wrong name and you would like to change it. One way to do so is to use the command "Update (file) Renamed newname<OK>". It works as follows: 2A5

The current file (and any existing modifications) are merged exactly

as in the "Update New" command, but the newly created file is

assigned the new name specified in the Update Renamed command. 2A5A

The old file is left exactly as it was, but minus any modifications. If you really want to delete all references to the old name, you must then delete the old file. 2A5B

There is a good reason that the old file is not automatically deleted when the new file is made. You can keep a template file (for example, a form letter, a proposal template, a programming template, and so on), and whenever you need a new version, simply edit the template. When you are done with the edits, simply use Update Renamed, and give the new file a name indicating its personalization. For example, suppose you have a standard letter you send out to your clients who complain about the cockroaches. You might keep a template file named BUG-LETTER-TEMPLATE, and when you send it to Mr. Smith, you can edit the template, and then use Update Renamed and give the new name SMITH-BUG-LETTER. 2A5C

Another way to rename a file is to use the Rename command: Rename (file named) oldname<OK> (to be named) newname<OK>. It just renames your file along with any existing modifications, without doing any updating. 2A6

File names in commands 2B

Almost all of the commands in AUGMENT that refer to a file have the following feature: You are asked in the command for a name of a file, and you can either type the name or, if it is the file that happens to be on your screen at the time, simply type <OK>. The prompt is "OK/T/[M/A]:". "OK" means type <OK> for this file, and "T" means type the name of the file; you must type <OPT> if you want to mark or give the address of a file name. 2B1

If you have your screen broken into two or more file windows, then you must be sure that your cursor is in the window containing the appropriate file when you type the <OK> indicating "this file". (Breaking your screen into more than one file window is discussed later; see <023>.) 2B2

If you worked with older versions of AUGMENT, what this means is that you do not have to jump to a file before you print it or check its internal structure for consistency. 2B3

Show Directory/Copy Directory 2C

The Show Directory command has been discussed elsewhere (<TUTOR3, 0220>), but remember that it is available at any time to present a list of the files resident in your (or in someone else's) directory. 2C1

There is another command, "Copy Directory", which gives exactly the same information as the Show Directory command, except that the information is actually entered in your file as text that may be edited, deleted, and so on. Remember that this option is available. Sometimes, users find it useful to have their directory presented as the first thing they see in their initial file when they enter AUGMENT. The file names are displayed as links and can be marked using the Jump (to) Link command

(see <0473>). 2C2

The two commands just discussed actually offer a large number of choices. Try the Show Directory command now, and notice that after you have typed the <OK> to indicate that you want to see your own directory,

you are presented with the prompt "OK/C:". The <OK> just says to show the file names, with no additional information. Type a question mark, and you will notice that there are, in fact, a large number of choices available for other information that can be displayed about each file. Such things as the protection, the dates and times of the last write, and many others can be displayed. Rather than going into detail about all of them, why don't you just try some of them out? Also, see <01536>. 2C3

Show Status (for file) 2D

This command is extremely useful. It tells many important things about the file at a glance. Try it now, and look over the information. 2D1

First you find out whether the file is being modified, and if so, by whom. 2D2

Next, the default directory for links is presented. This means that if a link is entered as text, and marked in the Jump (to) Link command, and if it points to another file, that file will be assumed to be in the directory which is the default directory for links. You probably want to have this be your directory in most cases, but there are times (usually for user documentation) when you want something else. See also <01222>. 2D3

Next, you find out when the file was last updated (creation date for this version), and you find out some information about the size of the file, including the number of statements and the number of pages used (to find out what a page is, see <0357>). Finally, since AUGMENT files can have holes in them, you are shown the percentage of the available space you have used. If this number is low, you will be advised to do an "Update Compact" (see <0467>) to attempt to remedy the problem. 2D4

Disk space 2E

Storage space on the computer comes in chunks called "pages". One computer page has (for the technically minded) exactly 512 36-bit words. This is equivalent to 2560 characters, or approximately the amount of information that can be typed on an "average" 8 1/2 by 11 inch double-spaced sheet. This last statement is, of course, not precise, but it is a good rule of thumb. 2E1

It is impossible to be assigned less than a whole page by the computer. If your file had only one character in it, a whole page would be assigned, and you would pay for the whole thing. Consequently, when AUGMENT files get holes in them (see Update Compact <0467>), it is a good idea to compress them. 2E2

Each directory has an allocation, or a number of pages allowed. If you use more pages than this allocation, you will probably have to pay extra for them. It is a good idea to get rid of any files that are junk, by

deleting them, and to archive files (see <0352>) that you will not need for a while. 2E3

To find out about your total disk usage, use the command "Show Disk (space for) Directory <OK>". Try this now. It is different from the "Show Status (for file)" command in that it shows the total usage of all the files in your directory. 2E4

Archive/Retrieve 2F

At present, this is done using the Executive Archive and Interrogate commands. 2F1

Modifications 2G

In other places in these tutor files, reference has been made to the fact that at any point, all of the changes you have made to a file since the last update are not "permanent", in the sense that you can delete them and get back to the condition of the file as of the last update. 2G1

The command to get rid of the modifications is "Delete Modifications". If you accidentally delete the modifications, and want to restore them, there is another command, "Undelete Modifications", that can be issued, but you must do so before you log out, before you expunge your directory, and before any automatic system expunges take place. 2G2

The mechanism is as follows: The file that you see on your screen can in fact be a combination of two files. One is the original file as of the last update; the other is called a "modifications file", and represents all the modifications that have been made to the original file. The original file knows nothing about the modifications, except that there are some. Whenever any text is to be displayed on the screen, both the original file and the modifications file are checked to see which one contains the newest information. Deleting modifications simply amounts to deleting the modifications file and marking the original file as being unmodified. The deleted modifications file is just like any other deleted file, and will disappear during an expunge operation. 2G3

In AUGMENT, when you show your directory, the existence of a modifications file shows up only as a message "[Being Modified By SMITH (JOHN)]". AUGMENT knows about the relationship between a file and its modifications, and never makes a mistake. If you are working from the Executive level, it is possible to err, since the Executive considers the two files to be completely different. In the Executive, the name of the updated version might be FILE-NAME.AUG;16, and the name of a corresponding modifications might be $SMITH$FILE-NAME.PC;16. The extension "PC" comes from the old name for the modifications file, which was "partial copy". 2G4

The reasons for modifications files are many. Some of the more important reasons include: 2G5

The original file is modified only during Update commands. Because of this, it is usually only opened for reading, and it is much less likely to be damaged during system malfunctions. 2G5A

•

If you have modifications to a file, no one but you can see them. If someone else jumps to a file that you are modifying, all they see is the version as of the last update, and a message indicating that you are modifying the file. Thus, if you want to make changes to a public file, you can try various things to see how you like them, and when you are finally satisfied, you can update and make the changes public. There is no reason for the rest of the world to see your first attempt at changes to the file. 2G5B

The mechanism makes it simple for many people to work on the same file. If one person is modifying the file, that person has it "locked", and no one else can modify the file until he or she

releases it either by updating it or by deleting the modifications. 2G5C

Needless to say, there are some problems that arise because of this, and they are described below: 2G6

If someone else is modifying your file, you cannot edit it until they release it. You either need to log in as that person (if you know their password) and update the file, or ask them to update it for you. 2G6A

A slightly more serious problem can occur if someone modifies your file and then deletes the modifications from their directory (with an Executive command). Your main file will then think it has modifications, but will not be able to find them. In this case, you can use the "Update Renamed" command and give the file its own name, but with a higher version number. For example, suppose your file is named ABCD.AUG;15, and someone has a modifications on it, so that you cannot modify it. Use the command "Update (file) Renamed abcd.aug;16<OK>", and you can then edit the new version 16. 2G6B

Real problems can arise if the files are moved from computer to computer, or are copied using the Executive Copy command (even if the modifications files are copied as well). The mess can be untangled, but it is quite complicated -- your best bet is to ask Feedback for help. You will never have these problems if you update your files before copying them or moving them. UPDATE YOUR FILES REGULARLY. 2G6C

Flavors of update 2H

For an elementary discussion of the Update command, see <TUTOR3, 0230>. 2H1

All of the Update commands are methods of merging the modifications file with the main file. There are four types of Update commands, and they have slightly different uses. They are: 2H2

Update New -- This is the usual command, and it combines the modifications with the file and makes a new, unmodified version of the file. The old version is kept as a backup, and the previous backup version is deleted. This command is fast, and should be used almost all of the time. 2H2A

Update Old -- This is a very fast update which is dangerous in that it leaves no backup. It can be useful in emergencies when you must

update but have run out of disk space. It is not recommended for usual editing; stick with Update New. There is a special case where Update Old is required. In some cases, individual directories are set up with a directory protection that will not allow you to create a new file in them, but will allow you to modify an existing file. Update New (which you would like to do) creates a new file (leaving the old one as a backup), while Update Old simply modifies the old one. 2H2B

Update Compact -- AUGMENT files are really not packed solid with information. When you delete statements or text, some holes are left, and when you insert new information, it is stuffed into one of the holes. Of course, things do not fit exactly, and after much editing, there may be a large number of small holes scattered through the file, making the file require much more space than it might. Update Compact goes through the file and takes out the holes. It leaves a backup, just as Update New does. When you "Show Status"

(see <0450>), you will sometimes be informed that an Update Compact might be useful. The Update Compact command does take longer than any of the other Update commands, however. We recommend that you try to do an Update Compact to a given file after every four or five uses of Update New. An Update Compact is also useful immediately after you have deleted a lot of material from a file. 2H2C

Update Renamed -- This is described elsewhere. See <0442>. 2H2D

Bad files 2I

Sometimes when you are dealing with a file, editing does not seem to work properly, and you may get an error message like "Bad statement identifier", or "blah blah blah -- file may be bad". At this point, you should issue the command "Check File (structure for internal consistency for file) <OK>". This will check through the structure of the file and make sure that it is OK. If not, a message "Bad File" will appear. If everything is OK, you will get a message like "Successful -- internal structure is OK". If you are a beginner, and you get the former message, don't do anything else with the file; call your representative or Feedback for help. If you are more advanced and want to try to fix it, read on. 2I1

Files can go bad because of a number of things. The most common include a system crash while doing an update, problems with an experimental program, and disk read errors, though there are others. Almost always, the error will be in a file opened for writing; consequently, it is almost always the modifications file that is bad, unless you are in the middle of an update when something wrong happens. If you have not done too much work since your last update, you can probably just delete the modifications, and everything will be OK. This should be another encouragement to update often, and to check your file often. 2I2

Another thing that often fixes bad files is an "Update Compact". This causes the file to be physically moved to another point on the disk, and sometimes this action magically fixes a bad file. You can try this second. 2I3

If this fails, you may have a very bad file. There are a number of

courses open to you that you can try. The more experience you have, the better luck you will have at fixing the file. Try such things as copying chunks out of the file to a new file, and verifying the new file after each copy. If the file goes bad, delete the modifications, and repeat the procedure, copying smaller pieces of the file. Continue until you have gotten everything that you can. Often, there will be one bad statement, and you will not be able to get to statements beyond that statement using the usual Jump to statement number commands. Jump to SID and Jump to Name will sometimes get you past the bad statement. 2I4

You can sometimes find out the exact statement where the file went bad by printing the file. The print will fail on the bad statement. 2I5

Another alternative is to do an "Create Sequential" with viewspec "y" on, and then a "Create Augment" to a new file, using Two returns between statements. (Create Sequential and Create Augment are described in <01496> and <01327>, respectively.) 2I6

Sometimes nothing you can do will help, and you must go back to a previous version. Check this file, and if it is good, do an Update Renamed to a version number higher than the bad file. If the backup

version is bad, you may have to go back to a dump tape. If the file is very important, call your Tymshare representative for help, or contact Feedback. 2I7

As is stated elsewhere in this file, AUGMENT files are not organized sequentially; new statements are just stuffed in anywhere, and pointers are used to show how to get from one statement to the next. 2I8

If the pointers get fouled up, AUGMENT may not be able to find the next statement, and a bad file results. In this case, a large portion of the file may be lost (unless it can be accessed via SIDs or statement names, which are not done with pointers). Sometimes, only a pointer to some text is lost, in which case only the text of the statement may be lost. The more you know about the internal structure of AUGMENT files, the better the chance that you will be able to fix a bad one. There is a good discussion of AUGMENT file structure in the programmer's guide, <USERGUIDES, PROGRAMMERS-GUIDE, 6d2>, which you might be interested in looking at if you are an experienced computer programmer. 2I9

Dump tapes/Backup 2J

Occasionally, no matter how good you are, you will make a mistake, and delete a file that you did not want to. If you merely delete it, you can undelete it, if you have not yet used the Expunge command or logged out. Sometimes, however, you will not give the Undelete command soon enough, and the file will be gone. You have the following options: (1) You can go to your backup version, if you simply deleted the highest version of your file. This will let you go back to the state of the file two updates ago. (2) Every night, a "dump tape" is made, and all the information on the computer is saved on temporary tapes. You can ask to have your file restored from these tapes. Contact Feedback, and include as much of the following information as possible: The full file name and version number; the last date it was known to be there; whether the file was being modified. You will lose any work done during the day, but this is usually a lot better than nothing. Let us know as soon

as possible -- the dump tapes are kept for only a month or so. 2J1

Wild cards "*" 2K

In the Delete File command, a whole class of files can be referred to by using an asterisk instead of a part of the file name. For example, "Delete File (named) test.*<OK> <OK>" would delete TEST.AUG, TEST.TXT, TEST.REL, etc. "Delete File (named) *.txt<OK> <OK>" will delete all your files with extension TXT. To undelete all deleted files, you might do "Undelete File (named) *.*<OK>". 2K1

The "*" wild card can be used when describing the version number as well. 2K2

Shortcuts 3

Quick viewspecs 3A

At this point, you know of two ways to change your viewspecs: You can use the "Set Viewspecs" command, or you can type viewspecs in response to the "V:" prompt that appears in most of the "Jump" commands. Often, you don't want to go anywhere, but you would like to make a quick change in the viewspecs. This can be done using the mouse. Hold down the left and middle mouse buttons, and type all the new viewspecs that you would like to take effect. When you are done, type one more special viewspec

(still holding down the two buttons). The special viewspec is "f". 3A1

Try this now, perhaps turning on (or off) numbers, blank lines, etc. Remember the final "f". 3A2

The reason that a final "f" viewspec is required is to let AUGMENT know that you are finished typing the viewspecs. The idea is that you often want to change more than one viewspec at a time, and it would be quite annoying if you had to watch the screen refresh each time a new viewspec was typed. The "f" (or "F") viewspec signals the end of a sequence of viewspecs typed using the mouse buttons. 3A3

The left and middle mouse buttons designate lowercase viewspecs. If all three mouse buttons are held down while a character is typed, the character will be interpreted as an uppercase viewspec. Try turning on (or off) SIDs using only the mouse buttons and not the shift key. 3A4

Insert mode 3B

Often, when you are first entering information into a file, you will find it inconvenient to have to type "is" (for "Insert Statement") before typing in each new paragraph. What you would like is a method by which you could enter paragraph after paragraph, each following the last one. AUGMENT allows for this with what is known as "insert mode". 3B1

Insert mode can be entered by pressing the INSERT key on your AUGMENT 1200 terminal. From that point on, you need only type paragraph after paragraph, and each will be inserted in succession. Before you type each such paragraph, you may give a level adjustment, which will indicate the level of the paragraph about to be typed (relative to the last one typed). To get out of insert mode, just use a command delete

as usual. 3B2

There is really only one question: Where does the first statement go? The answer to this is approximately the following: It goes after the last statement you "touched". Touching includes such things as jumping to or editing. It is a complicated concept to define in the case of a move or transposition, however. In any case, if you have no idea which statement you last touched, touch one, perhaps by using the "Insert Statement" command once. In other words, insert the first statement of the series in the usual way, and then just after you have finished, press the INSERT key, and you are off. (For experts: This can be done by simply pressing the INSERT key instead of typing <OK> at the end of the first Insert Statement command). 3B3

Go ahead now, and try to insert a few (short) statements following this one. Remember that if this statement is not at the top of your screen, it is probably not the last statement you touched. 3B4

The discussion of which statement you last touched may seem a little strange, but it makes a lot more sense when thought of in the context of using AUGMENT in typewriter mode. In that mode, it is extremely important to know exactly where you are, as all the editing commands will occur there unless you specify otherwise. 3B5

Repeat commands 3C

Any AUGMENT command may be repeated simply by pressing the REPEAT CMD key at the end of it. All command words up until the user needs to specify some content or a MARK or address are automatically repeated,

and continue to repeat until the CMD DELETE key is pressed. Try out this feature now with the "Delete Character" command. After marking the character to be deleted, press the REPEAT CMD key instead of typing the final <OK>. When you are in repeat mode, you need not continue to use the REPEAT CMD key -- <OK> works just as well. 3C1

It is very similar to the INSERT mode, and could be used on the Insert Statement command to give similar results, except that you would always need to mark the statement that the new statement was to follow. 3C2

Escape recognition 3D

The escape key (marked ESC or ALT on most terminals) provides a handy way to avoid a lot of typing. When you need to specify a file name, such as in a "Jump (to) Link" command, you do not need to type the entire name; simply type enough characters so that no other file in your directory begins with those characters, and press the <ESC> key. When you do this, AUGMENT will fill in the rest of the name, as if you had typed it yourself. If no such file exists, or if more than one such file exists, a "$" will appear. If you have not typed enough, backspace (to erase the "$"), and type a little more, followed by another <ESC>, if you wish. 3D1

Unfortunately, I do not know of any files that you have in your directory, except for TUTOR, TUTOR2, and TUTOR3, but they all must be typed to the bitter end before they are distinguishable. If you know of any other file in your directory, type a few characters at the

beginning, and try out <ESC>. 3D2

There are a couple of tricks that make use of the <ESC> key to make life a little easier. 3D3

Suppose that you have two files with names ABCDEFGHIJKLMNOP.AUG and ABCDEFGHIJKLMNOP.TXT. To delete the one with extension TXT, simply type "Delete File abc<ESC>t<ESC><OK>". The first <ESC> will do the best it can, filling out the name up to the period before the extension, and let you continue typing. The "t" is enough to distinguish between "TXT" and "AUG", so a second escape fills out the complete name. 3D3A

Suppose that you have made a serious mistake and destroyed some important information in a file, and you have also updated it. You decide that what you need to do is jump to the next lower version of the file and take a look at it to decide whether it is easier to start from there or to try to repair the newest version. It is likely that you don't remember any of the version numbers anyway, so use the following trick: Type "Jump (to) Link filename<ESC>". The name will be filled out to the end, including the highest version number. You can then backspace over the version number and type the number that is one lower. Be sure to retype the final comma, however. 3D3B

The <ESC> key can also be used to fill out directory names. Try typing the following (without the final <OK>!), watching the command window as you do: "jluserg<ESC>loc<ESC>". Notice that the directory name USERGUIDES was filled out first, and the second escape filled out the file name LOCATOR. Type a command delete if you do not really want to go there. 3D4

This brings up an interesting point: It is a good idea to give your

files names that differ in the early characters, rather than in the later characters. The pair ACCOUNTS-PAYABLE-79 and ACCOUNTS-PAYABLE-80 is very bad compared to the pair 79-ACCOUNTS-PAYABLE and 80-ACCOUNTS-PAYABLE. The <ESC> key is virtually useless in the former case. 3D5

If you are using AUGMENT in display mode on a nonstandard terminal using cursor keys, you will remember that typing an <ESC> will put you into pointer mode, rather than file name recognition mode. If you want to use <ESC> for file name recognition, simply precede the <ESC> with <LIT> (see <01260>). 3D6

Protection 4

There are two different kinds of protection that can be put on any file. AUGMENT protection allows you to protect a file so that only certain people can look at it. The people to whom you wish to allow access can be specified on an ident by ident basis. AUGMENT protection is an all or nothing thing -- you cannot allow someone the ability to read but not write, and so on. Executive protection allows a more general type of protection, but it is not on an individual by individual basis. For any given file, you can set "read but not write" protection, but only for certain groups of people; you cannot control it on an individual basis.

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 11

The two forms of protection can be used together, however. In order to view (or write on) a file, one must pass both protection tests. 4A

By using both AUGMENT and Executive protection on a file, you can get almost any effect you might desire. 4B

AUGMENT protection 4C

In order to make a file AUGMENT private, you must do three things. First, you must insert an AccessList of idents allowed to view the file, then you must set the file private, and finally, you must update the file. The three actions are discussed in detail below. 4C1

The AccessList: 4C1A

Somewhere in the origin statement of your file (probably immediately following the four semicolons), insert as text the following: "AccessList ID1 ID2 ID3 ... IDn;", where the capitalization of AccessList is exactly what is shown, and ID1, ..., IDn are the idents of the people to whom you wish to allow access. Be sure to include your own ident, or you will never be able to see the file again! Separate the idents with spaces, and end the list with a semicolon. Later, if you want to change the AccessList, you may simply add or delete idents from the list. 4C1A1

Setting the file private: 4C1B

Next, you must issue the command: "Set Protection (mode for file) <OK> (protection type:) Augment On <OK>". This tells AUGMENT that the AccessList is in force. If this command is not issued, the file is still not protected. To undo this command, making the file public again, use the command "Set Protection (mode for file) <OK> (protection type:) Augment Off <OK>". If you are interested in finding out the protection status of your file, use the command "Show Protection (for file)". 4C1B1

In both of the commands above, after you are prompted with "(mode

for file)", you may type a file name. The <OK> in the above examples means "this file". 4C1B2

There is a shortcut that allows you both to install an AccessList and to set the file private. Instead of typing "On" or "Off" in the command above, use the command word "Accesslist". You will then be asked for a list of idents. An appropriate AccessList is constructed (obviating the need for you to remember the exact form of the AccessList), and the file is set AUGMENT private. 4C1B3

Update the file: 4C1C

Before you do this, your changes to the AccessList and to the protection setting are modifications to the file, which other users will not be able to see. You certainly want them to "see" these modifications. Hence, you must update your file to have the protection truly take effect. In fact, since AUGMENT keeps two versions of your file available (usually), if you only update once, the most recent version will be protected, and the older

version will be unprotected. An "Update Compact" will get rid of the last unprotected version. Obviously, if you set the protection of a newly created file the first thing, you do not need to worry about this. 4C1C1

Executive protection 4D

From the point of view of the Executive, there are three classes of people: You yourself make up the first class, your "group" (to be described later) makes up the second class, and everyone else in the universe makes up the third class. You can set the protection differently (for any particular file) for each of the three classes. 4D1

"Your group" can be any specified set of Executive directories. If you want to set up such a group, you must send a message to Tymshare (or Feedback) requesting it. If you say nothing, your group will consist of only one person -- you. Often, your group will be set to be all the other people with whom you work. It is possible to be a member of more than one group, but there is no way of setting the protection differently for the two different groups -- either both can do such and such, or both cannot. If you wish to set protection on an individual basis like this, use AUGMENT protection, as described above. 4D2

For each of the classes of people discussed above, there are five individual types of protection that can be specified (for each file). You can allow individuals in the class to (1) read the file, (2) write on the file, (3) execute the file (if it is a program), (4) append to the end of the file, and (5) list the file in a Show Directory command. For AUGMENT files, you will probably not ever need to worry about the execute and append protections -- they refer mostly to non-AUGMENT files. 4D3

The list protection, when set, will not allow other users even to see that the file exists. The read and write protections mean the obvious things. 4D4

For each of the classes of individuals mentioned above, all five types of protection are coded up into a two-digit number, and your total protection is specified by putting the three two-digit numbers together to form a six-digit number. The details of the coding will be discussed later (and can be skipped by all but the most inquisitive). A few

simple examples should give you the idea of what you need to do. 4D5

The two-digit code "77" means that anything can be done to the file -- read, write, append, list, and execute. The code "52" allows all nonmodification commands to be done -- read, list, and execute. The code "00" allows absolutely no access rights to the file. 4D6

When the two-digit numbers are packed together, your protection comes first, then the protection relative to your group, and finally, the protection for everyone else in the universe. A very common protection is the following: "775200". This allows you to do anything, members of your group to look but not touch, and others to not even look. If you want to make a file that is completely secret to the rest of the universe, but which members of your group can write on, the protection should be set to "777700". To make a file so that you are the only

person who even knows of its existence, use the protection "770000". To set a file so that anyone in the universe can read it (but not write), and so that you are the only person who can write on it, use the protection code "775252". These examples should give you the idea. 4D7

Now, how do you go about setting the protection, assuming that you know the appropriate combination of digits? The command is the following: "Set Protection (mode for file) <OK> (protection type:) Exec Code (in octal) NUMBER<OK>", where NUMBER is the six-digit protection code described above. As was the case with AUGMENT protection, after the noise words "(mode for file)", you may type a file name, or type <OK> to mean the current file. Unlike the AUGMENT protection, Executive protection takes effect immediately. 4D8

If you do not want to remember all the details about the code, it is possible to set the Executive protection one item at a time. Instead of typing the command word "Code", you may type a sequence of command words of the form {Permit/Restrict} {Read/Write/Append/List/Execute/All} (access for) {Self/Group/Others} (finished?) ANSWER. You may type as many sequences as you wish, followed by "y" for "yes" or <OK> to end the command. The braces above mean that you can select exactly one of the alternatives contained within. An example follows. 4D9

Suppose you want to set the protection of the current file so that members of your group can read and write on it. Give the following command: 4D10

> Set Protection (mode for file) <OK> (protection type:) Exec Permit Read (access for) Group (finished?) no Permit Write (access for) Group (finished?) yes 4D10A

Now, for those brave souls who want to know the details of Executive protection, here they are. To set a particular protection, proceed as follows: For each of the classes of individuals, and for each of the five protection types listed above, write down, in order, a "1" or a "0", depending upon whether you want to allow or forbid that right, respectively. For example, if you want to allow everything, your list should be made up of five "1"'s. List the five digits in the order mentioned above: read, write, execute, list, and append. Finally, add a "0" or "1" to the end of your list to make six digits. This number is not used, but must be there. It makes no difference what you put down. 4D11

Next, break down the group of six digits into two groups of three, and convert the sets of three zeros and ones to digits using the following table: 4D12

000 = 0 4D12A

001 = 1 4D12B

010 = 2 4D12C

011 = 3 4D12D

100 = 4 4D12E

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 14

101 = 5 4D12F

110 = 6 4D12G

111 = 7 4D12H

The two digits you get will be the protection for one of the classes of users described above. The following example should illustrate the procedure. You want to be able to do anything, you want your group to be able to read, list, and execute, and you want the rest of the universe to be able to do nothing. You should get the following sets of digits (where the last of the six in each case is random, and could be changed): You, 111 111; your group, 101 010; and the rest of the universe, 000 000. Using the table above, these combinations translate to: You, 77; your group, 52; and the universe, 00. Hence, the entire protection for this file will be: 775200. 4D13

Note that because of the ambiguity of the sixth "0" or "1", the protections 775200, 775201, 765301, ... are all equivalent. 4D14

In addition to the types of protection discussed above, there is still another, more arcane type called "directory protection". It has to do with entire directories. See an Executive user's manual for a discussion of this type of protection. 4D15

There is one final question that needs to be answered, and then the discussion of protection will be complete. What determines the protection of a newly created file? The answer is that each directory has a default protection associated with it, and each new file gets that protection. If you need a directory in which newly created files automatically need some special kind of protection, it must be set up that way by Tymshare. Contact your representative or Feedback to have this done. (You may, for example, want to have a very private directory, because all the material in it will be relatively confidential, or you may want to have a directory in which all information is public, as, for example, a USERGUIDES directory). If you don't say anything when you ask for a new directory, the protection automatically assigned is "775200". You can do anything, your group can read but not write, and the rest of the universe can do nothing. 4D16

Advanced editing 5

Visibles and Invisibles 5A

Visible and Invisible are just two more nouns, similar to Text, Character, and Word. They are most similar to Word, in that they require one mark, and specify (possibly) more than one character. A visible is defined to be the marked position and as many characters to the right and left until a "nonprinting" character appears.

"Nonprinting" means a character like <SP>, <TAB>, <RET>, and many others. A Visible can be thought of as a sequence of characters that are visible on the screen. 5A1

A "Visible" can be an extremely useful entity -- it will grab attached quotation marks, punctuation marks, and so on. The following file name: TRAINING,TUTOR3.AUG;2 can be identified as a visible using one mark

anywhere within it. (Delete it now, with the "Delete Visible" command.) 5A2

Like the "Word" noun, the "Visible" noun can specify two adjacent visibles. Note that in the definition, it was stated that a visible is the character marked (whatever it may be) and ... Try moving, deleting, copying, etc. a few visibles in this statement until you understand fully what it means. Then, try to figure out what an invisible is before reading the next paragraph. 5A3

An "Invisible" is just the opposite of a "Visible". It is defined to be the character marked, plus any number of nonprinting characters to the right and left. Note that like "Word" and "Visible", an "Invisible" can contain one character that would normally not be a part of it -- the marked character. Delete the invisible between this sentence and the last one now, using "Delete Invisible". & Now try to delete the junk between this sentence and the last, using the Delete Invisible command. (Hint: Mark the "&"). 5A4

Formatting using invisibles 5B

This section begins with a disclaimer. If you need to do a lot of formatting, you should probably use the Output Processor or the Table subsystem. If you just need to make one or two statements look different, this can often be accomplished using return characters and/or tabs. 5B1

One of AUGMENT's great advantages is that there is no need, when you are typing text, to pay attention to where the lines must break. If you insert or delete a word, all of the lines in the corresponding paragraph are adjusted to make the line breaks occur as uniformly as possible. Because of this, statements can be freely moved to a lower or higher level, where a different indenting might make the lines not fit. The only time this is a problem is where you want to force a new line to begin. If you want it to begin a new line because it is a new paragraph, there is no problem -- just make a separate statement of it. There are times, however, when you want to begin a new line in the middle of a statement, as, for example, in a small table. 5B2

The text of the next statement is contained between the dashed lines. Put that statement at the top of your screen, and follow the instructions in the statement below it. 5B3

A Poem

There once was a poet from Peru,
Whose limericks end on line two.

--- 5B3A

There is a return character <RET> at the end of each of the lines (except the last line of dashes), and there is a <TAB> just before "A Poem". The easiest way to see these invisibles is to give some command like "Delete Character", and MARK where the character would be. Then

type a command delete (assuming you don't want to delete the character). Try marking spaces, the characters just after the ends of the lines, and where the first character would be in the line containing "A Poem".

•

Note the way that the highlighted characters appear. Using this technique, you can tell the difference between the various kinds of formatting invisibles. MARK the beginning of the line after "A Poem". This is how two <RET>s in a row appear. Now try deleting and inserting <RET>s and <TAB>s to get a feeling of how they behave. Be sure to try putting in more than one <RET> or <TAB> in a row. 5B4

Return characters are simple; they say: "Go to the beginning of the next line, no matter what". Tabs are more complicated. Where are the tab stops, how do you change the settings? What happens to <TAB>-formatted material when it is moved down a level? 5B5

It may even turn out that your user profile is set up so that you do not insert <TAB>s at all -- the appropriate number of spaces are inserted. For this reason, and all of the above, tabs deserve a section all of their own. 5B6

Tabs 5C

Introduction 5C1

When you are typing on an ordinary typewriter and you strike the <TAB> key, the carriage advances to the next tab stop, and the next character you type is printed there. If you strike the key twice, you are advanced two tab stops, and so on. On the final printed paper produced, however, there is no way to tell whether the carriage was advanced using the <TAB> key or by typing the right number of spaces -- both show up as blank space. In AUGMENT, the situation can be somewhat different. In the simplest case, AUGMENT behaves exactly as would a typewriter, and that is what we shall describe first. 5C1A

Spaces for tabs 5C2

In your User Profile, if you have spaces for tabs turned on, this means that when you press the <TAB> key during normal text entry, the appropriate number of spaces are inserted to start the next character at the next tab stop. The tab stops are also set with the Set Profile command, and changing them will not affect any tabbed material typed, since all the <TAB>s were immediately turned into spaces. If you have spaces for tabs turned on, and for some reason you want to put a real <TAB> character (see below) in your file, precede it with a <LIT> or <CTRL-V> character (see <01260>). 5C2A

The method outlined here is probably the simplest method for dealing with tabs, but it does have the disadvantage of having the tabs immediately "frozen" into spaces as soon as they are typed. The next section discusses alternative methods for dealing with tabs. 5C2B

Inserting real <TAB> characters in your file 5C3

In AUGMENT, there is a character called <TAB> or <CTRL-I> which can be inserted into your file just like letters, spaces, or return characters. It takes up the same amount of memory as any other character in the file, except that it has a special formatting effect on your screen and in printed output. 5C3A

•

When a statement is being presented on your screen or is being printed, and a real <TAB> character is encountered, the next character is presented at the next tab stop. The tab stops are set in the Set Profile command. 5C3B

If you have real <TAB> characters in your file, and you change the tab settings via Set Profile, then the screen will be presented differently. This can be both an advantage and a disadvantage. If you have text typed in with <TAB> characters, and it doesn't look right, you may just need to change your User Profile settings to improve how it looks, without retyping anything. On the other hand, if you want others to view your file on their screens, their tab stops in their User Profile will need to be the same as yours. In fact, if you change your own tab stop settings, you will not view it "correctly". 5C3C

Occasionally, using <TAB> characters can result in a great space saving in your file. To take a somewhat extreme example, suppose that you want to list information in two columns that are 50 characters apart, but most of the things in the first column are only 4 or 5 characters long. If that space is filled in with space characters, it will take 45 or so of them per line; if a single <TAB> is used with the right setting, only one <TAB> character will be needed per line. If the file is hundreds of lines long, this can amount to a considerable savings. 5C3D

Numbers 5D

A Number is just another editing entity, similar to Word, Character, Text, and so on. In most cases, it behaves exactly as you would expect. For example, try deleting this number: 137.04 using the Delete Number command. Insert Number, Move, Replace, Copy, and so on, all work. 5D1

It turns out that there are times when the number entity does not behave exactly as you might expect. The reason is that to define what a number is in terms of text is very difficult. Consider the following list of things, which could all be considered to be numbers under certain circumstances: "12", "12.34", "1,234,432.5", "1.2E23", "-56E+7", "+6.02E23", "(456)", "$54.12", "- 123", ".0000003", "1.6E-19", "E-7", "E3", "775200B". Not all of the above are recognized as numbers, but many of them are. Thus, sometimes slightly surprising things happen when you move a number -- not all of it is moved, too much is moved, etc. AUGMENT just makes its best guess as to what you mean, and it is not always right. Very often, the Visible noun works better than the Number noun. 5D2

There are times when the Number noun is quite useful. One is in table-like applications. Suppose that only three-digit numbers are allowed in a particular table, as in the tiny table below. Try using the Replace Number command on various of the entries below and watch what happens. Be sure to try to replace short numbers by longer ones, and long ones by shorter ones. Be sure that all numbers are less than or equal to three characters in length in this example. 5D3

123 34 993 567 5D3A

2 32 18 12 5D3B

(By the way, in the example above, there is nothing special going on. There are no <TAB> characters, and the numbers are just separated by the appropriate number of spaces. The point of the exercise is that the Replace Number command does some work to preserve spacing.) 5D4

Phrase 5E

A Phrase in AUGMENT is a textual entity that behaves something like Text. You can Delete, Move, Copy, ... a Phrase, and a Phrase is simply defined to be text that begins and ends with a visible. To identify a phrase requires two marks. A typical use would be to move a group of words, or a sentence. Phrase is useful in the same way that the Word or Visible entities are useful -- your marking need not be quite so precise. All you need to do is mark a character somewhere in the first and somewhere in the last visible required. The fact that a Phrase goes from visible to visible instead of word to word is that it is usually desired to carry along the punctuation, be it periods, quotation marks, or parentheses. 5E1

Directive 5F

The Directive noun refers to Output Processor directives. It is something like the Word noun, in that it requires only one mark to identify the directive in question. A directive is defined to go from a period on the left to a semicolon on the right. You must mark the period or semicolon or somewhere inside the directive in order to have it recognized. 5F1

Frozen statements 5G

Any statement (or statements) can be frozen, in the sense that it (they) will always be presented at the top of your screen, no matter where you may jump (even to other files). This is sometimes very convenient when you want to keep a statement containing reference information and use it in other places. The information in the frozen statement can be copied by marking it, and statements can be inserted to follow it, simply by marking it when you are asked "(to follow)" in the Insert Statement command. 5G1

The command is simply "Freeze Statement (at) MARK <OK>", and the command to unfreeze is "Thaw Statement (at) MARK <OK>". During a particular AUGMENT session, a statement will remain frozen until it is thawed; however, the frozen statements may be turned on and off with a viewspec. They are normally off. After you have frozen a statement, you may see it by setting viewspec "o". To turn off the frozen statements, use viewspec "p". Try freezing a statement, and then use viewspec "o" to see what it looks like. Try freezing and thawing various statements, with various combinations of the "o" and "p" viewspecs. Try jumping around, and notice that the frozen statements remain fixed. 5G2

If you want to both freeze a statement and turn on (show) the frozen statements, simply type "o" when you are prompted for viewspecs. 5G3

There is a shortcut in the Thaw command: "Thaw All (frozen statements)"

does just what it says. 5G4

Statement names 5H

An advanced user will find that statement names provide one of the most useful and powerful ways of getting around in an AUGMENT file. In a

large file, you may have a number of places where you regularly go, and if you are like most people you will find it hard to remember that you kept your to-do list at statement 17a, your accounting information at 41, and so on. What is worse, if you add a new statement 1 to your file, suddenly your to-do list is at statement 18a, and your accounting information can be found at statement 42. You may recall that SIDs do not change, but still it is hard to remember numbers like 0188 and 0423. In a heavily used file, it may be worse -- the SIDs may begin to look like 013245 or 088724. 5H1

It is possible, and far more convenient, to give important statements a name by which you can refer to them. Good names for the two statements mentioned in the paragraph above might be "to-do", and "accounts". In this section, we shall discuss how to go about this. 5H2

Every statement potentially has a name, and if it has a name, the name is defined by the first few characters of the statement. A name must start with a letter of the alphabet, and can then contain any combination of letters, digits, and the special characters "-", "@", and "'". For example, the following are legal statement names: "mail", "accounts-80", "name1", "qwer@-t3" (although why someone would name a statement this way is a little mysterious), and "journal". 5H3

The following are not valid names, for the reasons indicated: 5H4

#43	"#" is not allowed in a statement name	5H4A
5names	statement names must begin with a letter	5H4B
name[3]	"[" and "]" are not allowed in a statement name	5H4C

Unless you do something special, the name of a statement is simply the first characters in a statement, if they form a valid name. The name of this statement, for example, is "unless". 5H5

5This statement has no name, since "5This" is not a valid name. 5H6

Note that the capitalization of a statement name makes no difference -- the statement two before this one begins with the characters "Unless", but its name could be referred to as "unless", "UNLESS", or even "UnLeSS". 5H7

Once a statement has been named, the name may be referred to as an address element in the same way that statement numbers or SIDs are used. For example, the following commands would be valid in this file: 5H8

Jump (to) Link unless<OK> 5H8A

Jump (to) Link tutor3,unless<OK> 5H8B

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 20

Jump (to) Predecessor (of) unless<OK> <OK> 5H8C

Delete Statement (at) unless<OK> (really?) <OK> 5H8D

Now it should be clear why all file names require a comma after them. Without the comma, they are interpreted as a statement name, and AUGMENT tries to find a statement with that name. Try typing "Jump (to) Link tutor<OK>" (no comma), and note that there will be an error message because no statement (in this file) has that name. 5H9

Note also that statement names can be ambiguous. In many files, for example, quite a few statements have the name "the", since many paragraphs begin with that word. If you try to "Jump (to) Link the<OK>", you will just get to one of them, and there is no way of telling which one will be reached. This may sound bad, but it does not usually cause trouble, and there are a number of ways of getting around a problem of this sort, should it be serious. 5H10

A command "Jump (to) Name" exists for looking at successive statements with the same name. This command comes in three forms: 5H11

Jump (to) Name First stname<OK> takes you to the first occurrence of a statement with name "stname" in the file. 5H11A

Jump (to) Name Next stname<OK> takes you to the next occurrence of a statement with the name "stname" in the file. 5H11B

Jump (to) Name Any stname<OK> is just like the Jump (to) Link command. It is faster than the other two, and can be used if you are sure that there is only one statement with the specified name in the file. 5H11C

(NOTE: There is a fourth form, "Jump (to) Name External stname <OK>". However, its use is limited to VERY advanced users and applications programmers. If you need information about this command, contact your Tymshare representative.) 5H11D

Earlier in this section, it was stated that "unless you do something special, the name will be the first characters of a statement". In some instances, you would like to make the naming of a statement a special thing, so that not every statement will have a name (as most of them do in this file, for example). To understand fully how to do this, it is necessary to understand exactly how a statement name is determined. 5H12

Every statement in every AUGMENT file has two characters associated with it which are known as its "name delimiters". If nothing special is done, those name delimiters are not present -- they are NULL and NULL. This means that anything at the beginning of a statement that looks like a name IS a name. If the left and right delimiters were "(" and ")", respectively, then in order for the statement to have a name, the first characters of the statement would have to be a name surrounded by the characters "(" and ")", as in the following statement: 5H12A

(blap) The name of this statement would be "blap", if its name delimiters were "(" for the left delimiter and ")" for the right

delimiter (which they are not). 5H12A1

Try a Jump (to) Link blap<OK> to see that there is no statement in this file by that name. Then use the (new) command "Set Name (delimiters in) Statement (at) MARK (left delimiter to be) (<OK> (right delimiter to be))<OK>" to change them, and then try a "Jump (to) Link blap<OK>" again. 5H12B

The above command can be used to set the name delimiters for a statement, group, branch, or plex, and the name delimiters can be set to any characters. The following statement has name "qwerty", since its name delimiters are "$" and "[". Check this with a Jump (to) Link qwerty<OK>. 5H12C

$qwerty[This statement has strange name delimiters (and a strange name, too!) 5H12C1

The name delimiters can be set back to NULL by using the NULL key on your terminal when a left or right delimiter is requested. The left may be NULL and right a character, or vice versa. 5H12D

There is a command "Show Name (delimiters for statement at) MARK <OK>" to find out what the current name delimiters are. Try it on a few statements. 5H12E

If a statement has name delimiters "(" and ")", the only way it can have a name is to have the first few characters (which must be a valid name) surrounded by "(" and ")". Changing the name delimiters in a statement may suddenly cause it to have a name, or not to have a name. 5H12F

When a new statement is inserted into a file, it automatically gets the same name delimiters as the statement it is inserted under. (Remember that the origin statement is over everything, so the very first statement you put into a newly created file gets the same name delimiters as the origin. If you want to have a special file where all statements have special name delimiters, when you first create the file, set the name delimiters of the newly created origin statement to be what you want, and from then on, every new statement will have those special name delimiters. (Alternatively, you can change all the delimiters for statements in a file by changing them for the branch beginning at the origin using the "Set Name (delimiters ...)" command.) 5H12G

Relative addressing 5I

There are times when you would like to jump not to the predecessor of a statement, but to the predecessor of the predecessor of the statement. You might want to jump back three statements from the top of the screen, and so on. To do these things using the commands you presently know would require two jumps in the case of the first example, and three in the case of the second. This section describes some shortcuts that can be used to combine two or more jumps into one. 5I1

It turns out that all the specialized Jump commands that you have learned up to this point are in fact special cases of the "Jump (to)

Link" command, relative to the top statement on your screen. For example, if you want to jump back one statement from the top statement on your screen, you could use the command "Jump (to) Link .b<OK>". The period indicates that the following character is to be taken as a relative addressing element. In particular, "b" (or "B") means "Back". 5I2

Note that this gives exactly the same result as "Jump (to) Back" and marking the top statement on your screen. There are other addressing elements similar to .b, and they are listed below. This is a complete list, and some of them will require further explanation later. For now, just read through the list to get a flavor of the available addressing elements. Come back later when you need to use them. 5I3

.b back 5I3A

.c next occurrence of content 5I3B

.d down 5I3C

.e end (of branch) 5I3D

.h head (of plex) 5I3E

.l find and take the link 5I3F

.n next statement 5I3G

.o origin 5I3H

.p predecessor 5I3I

.r return 5I3J

.rf return file 5I3K

.s successor 5I3L

.t tail (of plex) 5I3M

.u up 5I3N

.w next occurrence of word 5I3O

Now, for the nice part. To jump back two statements from the top of the screen, simply use the command "Jump (to) Link .2b<OK>" or "Jump (to) Link .bb<OK>". 5I4

To jump two predecessors from the statement at the top of the screen, use the command "Jump (to) Link .2p<OK>". To jump to the statement down from the predecessor of the statement at the top of the screen, use the command "Jump (to) Link .pd<OK>". To advance 100 statements in the file (again, counted from the top statement on the screen), use "Jump (to) Link .100n<OK>". A useful little command, no? 5I5

Any combination of the above addressing elements can be used in a link; we have seen some examples. Also, they can be appended to the ends of

other links, as follows: Suppose you would like to jump to the successor of the statement named "blap". You can do so with the command "Jump (to) Link blap.s<OK>". 5I6

Following is a short discussion of the meanings of the address elements that are probably not familiar to you. Obviously, not all of them are equally useful. 5I7

.c -- next occurrence of content. If you have specified via a Jump (to) Content Next command some content to be searched for, the .c says to repeat that search, beginning from the current location. 5I7A

.e -- end. This refers to the last statement in a branch (that is, the last statement before another statement appears at a level equal to or higher than the current statement). The very last statement in the file can be reached by means of a simple command: "Jump (to) Link .oe<OK>" (jump to the end of the branch under the origin of this file). It can be quite handy. 5I7B

.h -- head (of plex). This command finds the first statement in the plex identified by the current statement. The last statement is referred to as the tail (.t is the address element). 5I7C

.l -- find and take the link. This element assumes that you have typed, as text, a link in the present statement (see <0473>). If you know where a link is, but not what it points to, this command allows you to follow it. 5I7D

.r -- return. Similar to .rf, but in the current file. To jump to the previous view in the previous file, you could use the command "Jump (to) Link .rfr<OK>". 5I7E

.rf -- return file. This element takes you back one file in your file return ring. (If you want to go back two, you should use "Jump (to) Link .2rf<OK>"). To get to the origin of the last file you were in, you can use the command "Jump (to) Link .rfo<OK>". 5I7F

.t -- tail (of plex). Find the last branch in the current plex. This is the opposite of head (.h). 5I7G

.w -- next occurrence of word. Very similar to .c, but insists on finding the content as a word. For example, if the previous content search had been for "the", .c might take you to a statement containing "there", but .w would insist that the statement contain the word "the". 5I7H

Try out some of these addressing schemes. They do take practice, but as your skill increases, you will find that you use them more and more. 5I8

Jump (to) Address 5J

The "Jump (to) Address" command is just a slightly souped up version of the Jump (to) Link command. In the Jump (to) Link command, all addressing is done relative to the statement at the top of the screen. The Jump (to) Address command allows you to mark a starting place. It also lets you enter viewspecs separately. It is slightly more powerful

than the Jump (to) Link command, but it does take quite a bit more typing, so it is less useful. Play around with it if you are interested. 5J1

The Set, Reset, and Show commands 6

Background 6A

A large number of different capabilities are hidden away in these commands, and many of the most common of them have been described elsewhere. Some of the material in this section will repeat the information found elsewhere in the tutor files, but all of the capabilities of these three commands are documented here. Even if you think that you already understand a particular Set, Reset, or Show command, it is recommended that you read through its description anyway. Sometimes the version presented earlier covered only a simple case of the command. Where a command has been completely covered elsewhere, a link is given to the complete description. 6A1

In many cases, a Set command has a corresponding Reset command, which returns the parameter back to a default. The Show commands allow you to see the status of various things, and the things that can be shown often can be Set or Reset. There are exceptions to all of the above, however. The easiest thing to do is to quickly read through all of the descriptions below to get some idea of the parameters covered by these commands, and then use question mark liberally when you actually need to

use one of them. 6A2

As you read through the sections that follow (especially the Show command), try it out, and see the kinds of information presented. 6A3

Set 6B

Content 6B1

This command allows a user to type a content analyzer pattern, which is a sort of customized viewspec. There is a whole language for describing content analyzer patterns that is discussed in <TUTOR, TUTOR-CA-PATTERN,>. That document also describes the commands for turning on and off the custom viewspec. 6B1A

External 6B2

This command is used to associate an external names file with the current file. An external names file is the file to be searched for the appropriate name in the "Jump (to) Name External" command. 6B2A

Force 6B3

This command is used to set the force mode for the Force command. Normally, if you do nothing and then use the Force command on some text, all of the affected text will be changed to uppercase. The force mode can be set to Upper, Lower, First (letter upper), and Sentence. Once you have set the force mode using this command, it will remain that way for the rest of your session or until you change it using the Set Force command again. See <TUTOR3, 0198> for more

information. 6B3A

Link 6B4

The Set Link (default directory name ...) command allows a user to name the directory to be searched when typed links in the file are followed. See <01222>. 6B4A

Name 6B5

The Set Name (delimiters in) command allows you to change the name delimiters in any structure (Statement, Group, Branch, or Plex). See <0188> for a discussion of names and name delimiters. 6B5A

Print 6B6

The Set Print command allows a user to temporarily (for one AUGMENT session) set the printing device to be something other than the usual device. 6B6A

Protection 6B7

The Set Protection commands are used to set both the AUGMENT and Executive protection for a given file. A complete discussion of the various aspects of protection can be found in <0215>. 6B7A

Status 6B8

Normally, the status window is the top two lines of your screen, and is located to the left of the viewspecs window. All of the messages

from the Executive as well as some of the messages from the AUGMENT system are presented in the status window. At times, it might be useful to have a larger status window. The Set Status (window ...) command allows you to specify another window on your screen to temporarily be the status window. Use the Reset Status (window) command to put it back in the usual place. 6B8A

Temporary 6B9

The Set Temporary (status for modifications to this file) command allows you to appear to edit a file, without actually editing the file. This can sometimes be useful if you do not have write access to the file in question, and you would like to see how it would look with certain modifications. The modifications disappear as soon as the file is not on your screen any longer. 6B9A

Terminal 6B10

Sometimes it would be useful if your terminal were a typewriter terminal. You can set your terminal type to be typewriter using the Set Terminal (mode to be) Typewriter command. Set Terminal (mode to be) Display returns you to display mode. 6B10A

Viewspecs 6B11

This command changes the viewspecs. For a quick way to do this using

the mouse, see <0193>. For a discussion of all the viewspecs, see <09>. 6B11A

Profile 6B12

This command allows you to access a menu containing a whole host of parameters, called Profile Features, which you may set. These Profile Features replace the old Useroptions subsystem. A complete discussion of all the parameters you may set is beyond the scope of this document. For more information see <userguides,profile,>. 6B12A

Reset 6C

Force 6C1

The command Reset Force (mode) changes the force mode back to uppercase. See <TUTOR3, 0198> for a discussion of the various modes for the Force command. 6C1A

Link 6C2

This command changes the link default directory for the file back to the directory in which the file resides. See <0479> for a discussion of the default directory for links. 6C2A

Name 6C3

This command allows you to reset the name delimiters in the given structure to the defaults as specified in your user profile. See <0188> for a discussion of statement name delimiters. 6C3A

Print 6C4

This command undoes the the effects of the Set Print command. It

sets the default printing device for the session back to the usual device. 6C4A

Protection 6C5

This command can be used to reset either the AUGMENT or the Executive protection of a file. If AUGMENT protection is reset, it is turned off, allowing others to look at the file. Resetting the Executive protection changes it to whatever the default Executive protection for the directory is. See <0215> for a discussion of the details of both AUGMENT and Executive protection. 6C5A

Sids 6C6

The SIDs (statement identifiers) are the unchanging numbers associated with the statements in an AUGMENT file. The origin statement (the first to be created) has the SID 01. The next statement inserted in the file gets SID 02, the next, 03, and so on. These numbers are preserved, even when statements are moved around. If a statement is deleted, the SID is never used again. All of the links in this file use SIDs, because it is known that they will not change. The Reset SIDs command is the only way to change the SIDs in

a file. It renumbers the SIDs, starting from 01 as the origin. This is a very rarely used command. If the SIDs in this file were reset, for example, none of the links would point to the right place. 6C6A

Status 6C7

This command can be used after the Set Status command (see <01559>) has been used. It returns the status window to its usual place as the top two lines on the display screen. 6C7A

Temporary 6C8

This command removes you from temporary modifications mode (see <01553>). 6C8A

Terminal 6C9

This resets the terminal type to be whatever it was when you entered AUGMENT (see <01556>). 6C9A

Viewspecs 6C10

This command resets the viewspecs back to what your user profile says they should be when you enter AUGMENT. 6C10A

Profile 6C11

This command allows you to access a menu containing a whole host of parameters, called Profile Features, which you may reset. These Profile Features replace the old Useroptions subsystem. You may reset each profile feature one at a time with the Reset Profile command. A complete discussion of all the parameters you may reset is beyond the scope of this document. For more information see <userguides,profile,>. 6C11A

Show 6D

Changes 6D1

Occasionally, when you enter AUGMENT, an AUGMENT system message will be presented on your screen before the "BASE C:" prompt appears. After you issue your first command, this message disappears. Since it is presented only once, it is sometimes useful to be able to read it again. The Show Changes command allows you to do this. At present, only the most recently displayed presented message will be shown by this command. 6D1A

Content 6D2

This command shows the text of the last content analyzer pattern that was typed. See <TUTOR, TUTOR-CA-PATTERN,> for a complete discussion of content analyzer patterns. 6D2A

Directory 6D3

This command is simple in concept, but has many many subcommands. It

will show you a list of files in a directory (Show Directory <OK> shows your connected directory). The subcommands refer to the types of information that can be presented with each file name listed. Such things as author, date and time of write, protection, and others can be presented. In addition, the information can be sorted in a number of ways using the Sort subcommand. The easiest way to learn to use the command is using question mark. 6D3A

Exactly the same information can be entered as text in an AUGMENT file using the Copy Directory command. 6D3B

Try "Help dirtype<OK>" (use the Help comand and ask for "dirtype") for more information on the individual subcommands. There are about 60 of them, and most of them have an obvious meaning. 6D3C

Disk 6D4

This command has two forms. Show Disk (space for) Directory <OK> shows how much space is used in your directory. "Show Disk (space for) File amoeba<OK>" shows the size (in Executive pages) of the file named AMOEBA. 6D4A

External 6D5

This shows the file of external names that is to be searched when the Jump (to) Name External command is used. 6D5A

Link 6D6

This command shows in which directory the search for file names appearing in typed links is to be carried out. See <0479>. 6D6A

Marker 6D7

This command shows a complete list of markers for the file. This feature is not currently supported in AUGMENT, but was available in older versions of the system. When re-implemented, Markers will be called Tags. 6D7A

Modification 6D8

This command does a small part of what the Show Status command does

-- it tells you the modification status of the file named. 6D8A

Name 6D9

This command shows you the name delimiters associated with any given statement. See <0188> for a complete discussion of statement names. 6D9A

Protection 6D10

This command shows you all of the protection associated with the named file, both AUGMENT and Executive. See <0215> for a complete discussion of the protection features available in AUGMENT. 6D10A

Return 6D11

There are two commands under this: Show Return and Show Return File. The commands show you the return ring and and the return file ring, respectively. They show you where a Jump (to) Return or a Jump (to) Return File will take you. See <TUTOR3, 0139> for a discussion of the Jump (to) Return and Jump (to) Return File commands. 6D11A

Status 6D12

This command allows you to find a lot of general information about a file, including its protection status, its modification status, its creation date, and a number of measures of its size. 6D12A

Temporary 6D13

This command will allow you to find out if the modifications you are making to the current file are actual modifications or temporary modifications. See Set Temporary <01553>, described above. 6D13A

Version 6D14

From time to time, new versions of the AUGMENT system are released, and the Show Version command is a method of finding out exactly which version is running. If you are reporting a problem with the system, it is helpful if you include in your message the version number of the system involved. 6D14A

Viewspecs 6D15

This command shows all of the viewspecs currently in force. The command is rarely used, since you can usually tell what viewspecs are on, just by looking at your screen. Some things, like blanks lines between statements or numbers on, are obvious. All of the nonobvious viewspecs are presented in the viewspec window in the upper right corner of your screen. The only part of that display that is hard to interpret is which of the numbers corresponds to lines, and which to levels. An easy way to remember is that they are in alphabetical order: LEVELS LINES. 6D15A

Profile 6D16

This command allows you to access a menu containing a whole host of parameters, called Profile Features, which you may show. These Profile Features replace the old Useroptions subsystem. You may use this command to show the current setting of any of your Profile Features. A complete discussion of all the parameters you may show

is beyond the scope of this document. For more information see <userguides,profile,>. 6D16A

Miscellaneous AUGMENT commands 7

Background 7A

Most of the important AUGMENT commands have been discussed in detail elsewhere in these tutor files. There is a certain set of commands that do not fit nicely into larger categories, so they are all lumped

together here. Do not think that simply because the command appears in this section that it is not useful. The Check Mail command is very useful, for example. 7A1

Show Invisibles Y/N 7B

This command allows you to display spaces, carriage returns and tabs as tilde (~), open angle bracket (<) and closed angle bracket (>), respectively. The displayed characters will be shown on your screen, but will not be printed on output. This feature may be extremely useful in fixing up a Table, where the formatting spaces may be invisible, but important, or in determining if a file has been prepared with spaces for tabs set on or off. 7B1

Check Mail 7C

This command allows you to remain in AUGMENT and check your Executive mailbox for new mail. When the new mail system is installed, it will be used to check for the arrival of that kind of mail, too. 7C1

Clear (status window) 7D

The top two lines on your screen are the status window, and are used to present messages from the system. In the Conference subsystem, comments typed by participants who do not hold the gavel appear in the status window. It is possible (using the Set Status command) to make any other window the status window if a larger status window is needed for some reason. 7D1

Comment 7E

This command simply allows you to type characters that will appear on the screen; the characters will be erased as soon as an <OK> or <CD> is typed. It is useful in the Conference subsystem. This is a universal command, so it is available in all subsystems. 7E1

Mark (character at) 7F

This command is used to give a particular character in your file a name called a "marker". 7F1

Point (on shared display) 7G

This command is also available primarily for users of the Conference subsystem. It allows the user to mark a character on the shared screens, thus pointing at it. 7G1

Rename (file named) 7H

This command allows you to rename a file. It will not work at present

if the file to be renamed is the current file. 7H1

Reverse 7I

There are really two commands here. The first reverses the color of the background of a window (on an AUGMENT 1200 terminal), and the second

reverses the order of the statements in a Group or Plex. The first command is "Reverse Window (color value in window at) MARK <OK>", and the second is "Reverse {Group/Plex} (at) LOCATION <OK>". 7I1

Sort 7J

The AUGMENT Sort command allows you to sort a branch, group, or plex of statements into various orders. If you are sorting a branch, the statements down a level from the branch head are sorted. In the case of a group or plex, the statements in that group or plex are sorted. If the statements have substructure, the substructure remains connected to the statement above it, and is moved around wherever that statement goes. If you give the command in the simplest form, "Sort Plex (at) MARK <OK>", the sorting is in standard alphabetical order. See below for descriptions of all the available sorting types. More than one type can be used at the same time. A reverse chronological sort can be done using Sort ... Date Reverse <OK>, for example. 7J1

Sort STRUCTURE Alphabetic <OK> 7J1A

This is the default, described above. 7J1A1

Sort STRUCTURE Content <OK> 7J1B

If a content pattern is set, this command begins the sort from the character where the content pattern leaves off. See <TUTOR, TUTOR-CA-PATTERN,> for details about content patterns. If you understand something about content patterns, please read the statements below: 7J1B1

Suppose that you have a plex of a table where the first ten character positions are the person's name, and the next ten are a serial number. Suppose further that you would like to sort this plex on the serial numbers. To do so, set the following content analyzer pattern: 7J1B1A

10CH 7J1B1A1

Then use the command: 7J1B1B

Sort Plex (at) MARK Content (analyzer call) <OK> 7J1B1B1

This will do the trick. The content analyzer leaves the current character position after the tenth character, and that is where the sort will begin. 7J1B1C

There is another little-used feature that will allow you to end a sort after a given character position. Suppose that the example is the same as that presented above, except that there is some garbage in the character positions beyond the serial number which you do not want to include in the sort. In that case, you could use the following content pattern: 7J1B1D

10CH ^p2 10CH ^p1 p2 7J1B1D1

The global text pointer p1 is used to indicate the character

position at which the sort should terminate. p2 is another global text pointer that can be used as a temporary marker. In the example above, the content pattern says "Move 10 characters, and mark where you are temporarily using p2. Then move ten more characters, set p1, and reset the current position to p2". For content analyzer experts, the following pattern will also do the trick without the benefit of the temporary text pointer p2: 7J1B1E

20CH ^p1 < 10CH > 7J1B1E1

There are times, however, when both text pointers are needed. 7J1B1F

Sort STRUCTURE Date <OK> 7J1C

This command causes the sort to search through the text until it finds a valid date. The sort is carried out in chronological order. 7J1C1

Sort STRUCTURE Reverse <OK> 7J1D

The sense of the sort can be reversed with this command. Sorting by reverse alphabetic sorts the z's first, then the y's, and so on. 7J1D1

Type 7K

This command is primarily of use in typewriter AUGMENT, where it prints the appropriate text and then does a jump to that statement. In display AUGMENT, it prints the structure on the screen, and leaves it there until an <OK> is typed. Viewspecs can be specified in both cases. This command can be useful in display AUGMENT to have a sort of scrolling mode. You can read a large portion of text by typing a character after the end of each screenful without having to do jumps. 7K1

Single character commands 7L

Background 7L1

Most of the commands mentioned here are primarily useful for typewriter AUGMENT. All of them behave approximately the same way in display and typewriter mode, but there are some differences. The main one is that in typewriter mode, the commands are used for printing and moving in a file at the same time. 7L1A

<TAB> (repeat content search) 7L2

Typing this character repeats the last content search from the current position. The search type (content or word) is remembered as well as the text of the actual string. 7L2A

<LF> (line feed) 7L3

This command displays the text of the next statement. In typewriter AUGMENT, your current location is also moved to the next statement. In display mode, this command is virtually useless. 7L3A

. (period) 7L4

This command shows the statement number and character position of where you currently are. In typewriter mode, the character position makes sense; in display mode, it is always character 1 of the statement at the top of the screen. 7L4A

/ (current context) 7L5

This command can be used effectively in the middle of typing an address. It shows the current context. It can be very useful for typewriter mode "blind" edits. 7L5A

\ (current statement) 7L6

This command is just like "/" above, except that the entire statement pointed to is displayed. It is very useful in typewriter mode, and can be useful in display mode, as the following example shows. 7L6A

Suppose that you would like to move a statement from the current file into another file that is not on the screen (or to another place in this file that is not on the screen). Give the command "Move Statement", mark the statement to be moved, and then type the address (or your best guess) where it should go. Then, before typing the final <OK>, type "\". This will show the statement it is to follow. If you know your way around that file, and realize that you really want it to go following the successor, type ".s", and another "\" to see where you are. You can continue to adjust the address until it is exactly where you want it. When you are pointing to the right place, give the final <OK>. 7L6A1

; (comment) 7L7

This command allows you to type a comment without interrupting your work session. On a display, the entire display is cleared, making essentially a 24-line status window. The Comment command can also be used, and only enough of your display to present the comment will be overwritten. To get out of this semicolon command (or the Comment command, for that matter) type a command delete or <OK>. 7L7A

^ (previous statement) 7L8

This command does exactly the opposite of what <LF> does. In display and typewriter mode, it types out the content of the previous statement. In typewriter mode, it moves you to that statement in addition. Again, this command is not intended for use in display mode. 7L8A

Commands structures and the Process command 8

General introduction 8A

The basic idea of a commands structure is that there are occasions when you want to be able to give the exact same sequence of commands over and over again. For example, you may want to pull your messages in and

format them in a certain way, and jump somewhere, etc. Here is a sample commands branch. If you give the command "Process (commands from) Branch" and mark anywhere in the word "(sample)" you will see the system

jump automatically to the origin of this file and then take you back here. Do this now. 8A1

(sample) 8A1A

jump origin tutor4,<OK><OK> 8A1A1

jump return<OK><OK><OK> 8A1A2

Usually commands structures are used where you do some repetitive task, like when you always go through the same sequence of file commands. If you give them a name and "Process Branch" at that name it will actually do those commands automatically. Note that there is nothing special about a branch. You can also process commands in a Statement, Group, and Plex. Most commonly, however, they are organized as a branch. 8A2

Command recognition 8B

In a commands structure, you could type, for example, "G <OPT>" instead of "Goto <OPT>". It is best, however, to type the whole command word out. That way when you go back to the structure later on you can easily recognize the commands you specified. Also, you must type a space after every command word. 8B1

Inserting <OK> and other special characters 8C

You MUST insert the actual <OK> character. This is done by preceding it with <LIT> or <CTRL-V>. If you type <LIT> or <CTRL-V> first, then the next character you type will be literally inserted into the file. (<LIT> means take the next character literally.) Using <LIT>, you can get all of the characters you need in your commands structure. You can't get <CTRL-C> or <CTRL-T>, but any reasonable character can be inserted. In the sample commands branch in <01714> above, the actual characters typed in the first command statement were "jump origin tutor4,<LIT><OK><LIT><OK><OK>". 8C1

Creating a commands branch 8D

First of all, create a branch with an appropriate name enclosed in parentheses, such as "(mail)". Then just type the actual commands you want it to perform, using one statement for each complete command. For example: 8D1

(mail) 8D1A

Goto <OPT>message<OK> 8D1A1

Using parentheses as the name delimiters is recommended. You must set the delimiters with the Set Name (delimiters) command, as follows: 8D2

Set Name (delimiters in) Branch (at) MARK (right delimiter to be) (<OK> (left delimiter to be))<OK> 8D2A

The Process command will skip names without processing them. 8D3

Frequently it is helpful to issue your command sequence at a typewriter terminal before writing your commands branch. That way you can see your commands and exactly how they are to be specified. 8D4

Testing a commands branch 8E

Don't expect your commands branch to work the first time. You will probably find you left out a final <OK>, for example. Remember to update your files first because the commands may modify your files. 8E1

The commands being tested may go by so quickly that you won't be able to tell when something goes wrong. Therefore, you should first process a statement at a time, preferably at a typewriter terminal so you will have printed copy of what goes wrong, if anything. 8E2

Running a branch after testing 8F

The command is "Process (commands from) Branch (at) statementname<OK>". After you do this, AUGMENT performs the commands in the named sequence until it comes to the end of the branch and is done. 8F1

Running a commands branch from other files 8G

If you want to run the commands branch when you are in other files you must specify the file name, as in "Process (commands from) Branch (at) filename,statementname<OK>". 8G1

Startup command in Useroptions 8H

You can automatically run ONE commands branch every time you enter AUGMENT. The Useroptions subsystem command "Startup (Commands Branch Address)" will automatically process the commands in the indicated branch every time you enter AUGMENT. It is just a commands branch that could be run manually. 8H1

Hints 8I

Often you don't remember the exact sequence of a command. Use AUGMENT in typewriter mode and it will show exactly how many times you need <OK>'s, etc. 8I1

A commands structure is always much more complicated to run than you think. If you have done anything wrong, bad things can happen. 8I2

You may have filled up your program buffer and it can't load the program you request. So it is best to make the commands foolproof. Have one of your commands be: 8I3

Execute Programs 8I3A

Delete All <OK> 8I3B

Now you know you will have space. 8I4

•

Also: 8I5

Connect Directory yourname<OK>yourpassword<OK> 8I5A

Jump Link yourfile<OK> 8I5B

Have the commands do all the things necessary to make sure they are in the right directory and the right file. 8I6

Branches are only good for going through a list of instructions from beginning to end. They are not conditional; that is, they will not

execute certain commands depending on a condition. 8I7

Final notes 8J

You can't go to the Executive with a commands structure. 8J1

A commands structure that has been running for a long time may suddenly break down mysteriously. Check your commands structures periodically. For instance, let's say you wrote a branch that creates a new file in your directory each time it is run. Furthermore, let's assume that you have accidentally gotten very close to your disk allocation limit. When the branch gets to the place where it is to create the new file, it has no way of knowing that AUGMENT will not let it do so, and the branch will fail. 8J2

There is the basic question as to when you will use commands branches. While you are working, try to keep track of places where you seem to always issue the same sequence of commands over and over again. For example, if you always print out a document in a fairly funny way, where you have to set up a lot of things beforehand (e.g. set viewspecs, jump file, set up printer, etc.), and you always do these same ten things, put them in a commands branch. 8J3

Also, you may have a one-shot thing, where you want to replace the last character in every statement and it would take forever manually. Use a commands branch once and then delete it. 8J4

Remember the Useroptions way of setting up a Startup branch for every time you enter AUGMENT. 8J5

More on links 9

Links as text 9A

The cross-referencing in this file is an example of this. Any link that you can type in response to the "Jump (to) Link" command can be typed as text, and surrounded by "<" and ">", or by "(" and ")". Then, when you issue a "Jump (to) Link" command, you need only mark between the bracketing characters, and give the final <OK>, and you will be there. For example, <2> points to statement number 2 in the current file, and <TUTOR,2> points to statement number 2 in the TUTOR file. <09:wyIGm> points to the statement in this file that discusses other viewspecs, and says that when you arrive, all lines and levels are to be shown, blank lines are to be on, and the numbers are to be SIDs, placed on the right of the screen, and turned on. 9A1

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 37

A link can be thought of as a "view" of a file, and in this section, you will learn a large number of other things that can be typed either between the link bracketing characters or directly in response to the "Jump (to) Link" command. 9A2

Link defaults 9B

Every AUGMENT file has associated with it a default directory for links. This means that if you use the Jump (to) Link command and mark a link that doesn't specify a particular directory, the directory it will assume is the default directory for links for the file containing the link. 9B1

This is easiest to understand with an example. Suppose that whenever

you typed "Jump (to) Link" and marked a link, AUGMENT always assumed that you meant in your own directory, unless the link specifically requested something else. Then when you are looking at someone else's file, and they have put in a link, you cannot use it, unless they specifically included the directory name in the link. What really happens is that the file itself knows what directory the links are to assume, and this is what is used whenever a directory name is not mentioned and a link is marked as text rather than typed in the command. 9B2

Components of a link 9C

A link (whether typed in the "Jump (to) Link" command or typed as text) may consist of the following components:

directory-name,file-name,in-file-address:viewspecs;content-pattern;

Most of the components can be left out. If the directory name is included, the file name must be included as well, but anything else can be omitted. 9C1

Directory name 9C2

This is simply the directory name, which is used to log in. 9C2A

File name 9C3

This is the name of the file. It must be followed by a comma, but it may or may not include all of its parts (which include the extension and the version number). The following three examples should show the general idea: "BLAP" -- the file named BLAP, highest version or all versions, depending upon the command. "BLAP.AUG" -- same as BLAP. "BLAP.AUG;14" version 14 of BLAP, whether or not it is the highest version. 9C3A

In-file address 9C4

The in-file address is any combination of the address elements listed below. The address is evaluated from left to right, and all parts of the address must make sense for the link to be valid. After all of the possibilities are described, some examples will be shown. 9C4A

Statement names (see <0188>). A statement name is a valid address element. 9C4B

By preceding the statement name with the character "*", you refer to the next statement having that name. Preceding it with a "!" restricts the search to the current branch, and preceding the name with a "$" says that the name is to be interpreted as an external name. 9C4B1

Content 9C4C

To search for specified text, simply enclose it in double quotes. "abc" in a link is equivalent to "Jump (to) Content Next abc<OK>". If a single character is being searched for, you can shorten this to just a single quote followed by the character. For example, to search for the character "<", you could specify '< in a link. 9C4C1

Any content search can be restricted as follows: "abc"=w3s would insist that the content "abc" be found as a word, and that it be found within the next three statements. The expression "abc"=2w3s means that the second occurrence of the word "abc" within the next

three statements is sought. "abc"=4s will search for "abc" in the next four statements, but it can be either as text or as a word. 9C4C2

Statement numbers/SIDs. Any statement number or SID can be used as an address element. 9C4D

"Period" elements 9C4E

See <08>. Any of the address elements described there can be used in a link. 9C4E1

"Signed" elements 9C4F

The relative addressing elements described in the section on relative addressing (see <08>) all begin with a period and refer to other statements in a file. In fact, each refers to a particular character in the file -- namely, the first character in the statement. In display AUGMENT, you hardly ever need to worry about referring to particular characters within a statement, but it is possible, and is absolutely necessary when you are using AUGMENT in typewriter mode (see <0325>). 9C4F1

For completeness, these inter-statement pointers are described here, together with one or two (perhaps somewhat bizarre) examples of where they might be of interest to a display user. The following is a complete list: 9C4F2

c -- skip to character 9C4F2A

e -- skip to last character in statement 9C4F2B

f -- skip to first character in statement 9C4F2C

i -- skip to invisible 9C4F2D

l -- skip to link 9C4F2E

n -- skip to number 9C4F2F

v -- skip to visible 9C4F2G

w -- skip to word 9C4F2H

These address elements must be preceded by a "+" or "-" sign and, as was the case with the address elements preceded by a period, may be preceded by a number. Here are a few examples: 9C4F3

+4c -- skip ahead four characters 9C4F3A

-3w -- skip back three words 9C4F3B

+l -- skip to the beginning of the next link 9C4F3C

+e-1.1 -- find and take the last link in the statement 9C4F3D

Here are the bizarre examples I promised: 9C4F4

You have a statement on your screen with two links in it, and you would like to follow the second one but add your own viewspecs, say "my". Suppose the statement has SID 01234. You

can use the command: "Jump (to) Link 01234+2l.l:my<OK>". 9C4F4A

The spelling correcter program, when put into recording mode, will return a sequence of "misspellings" that looks something like this: (The numbers at the beginning of the statements are supposed to be statement numbers -- if you have yours on too, it will be confusing.) 9C4F4B

1A2A Misspelling for < DAVIS, TUTOR3.AUG;22, > 9C4F4B1

1A2B viewspecs < DAVIS, TUTOR3.AUG;22, 03+234c > 9C4F4B2

1A2C dut < DAVIS, TUTOR3.AUG;22, 03+414c > 9C4F4B3

1A2D Viewspec < DAVIS, TUTOR3.AUG;22, 03+741c > 9C4F4B4

1A2E connsequences < DAVIS, TUTOR3.AUG;22, 0233+349c > 9C4F4B5

1A2F uu < DAVIS, TUTOR3.AUG;22, 014+230c > 9C4F4B6

The "misspellings" mentioned in 1a2b, 1a2d, and 1a2f are probably not, but each of the others can be corrected in one step. Here are the appropriate commands: 9C4F4C

Replace Character (at) 1a2c.l<OK> (by) b<OK> 9C4F4C1

Delete Character (at) 1a2e.l+3c<OK> 9C4F4C2

I told you the examples were bizarre! These commands will not be used very much except in typewriter AUGMENT. 9C4F4D

Examples 9C4G

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 40

Find the first occurrence of the text "abc" in the last file you were in. 9C4G1

Jump Link .rfo "abc"<OK> 9C4G1A

Find the first occurrence of the WORD "abc" in the last file you were in. 9C4G2

Jump (to) Link .rfo "abc"=w<OK> 9C4G2A

Find the last statement in the last file you were in. 9C4G3

Jump (to) Link .rfoe<OK> 9C4G3A

You have a statement named "cook" in a file named LINKS which has a link in it pointing to a good online cookbook. Find the statement named "borscht" in that cookbook. 9C4G4

Jump (to) Link links,cook.l borscht<OK> 9C4G4A

In the origin statement, suppose that after the four semicolons, there is a link to something important. Get to the important thing. 9C4G5

Jump (to) Link .o ";;;;"+1.l<OK> 9C4G5A

Viewspecs 9C5

Following a colon, any viewspecs specified will take effect. 9C5A

Content pattern 9C6

This complicated and very powerful tool is discussed in another file, named <TRAINING, TUTOR-CA-PATTERN,>. 9C6A

Other viewspecs 10

This section attempts to discuss most of the viewspecs that were not discussed in other places in the tutor files. They are arranged in alphabetical order, not in order of conceptual difficulty. 10A

e -- The level of the referenced statement. This viewspec is extremely useful for clipping. If you jump to a statement using the "Jump (to) MARK" command, and set viewspec "e", you will see levels only down to the level of the statement you marked. The nice thing is that you do not need to count how many levels it is indented. "e" just says "the same level as this one". If you want to see statements at that level, plus those down one more level, type "Jump (to) MARK eb<OK>", so the viewspecs are "eb" -- that level, plus one more. 10A1

g -- Show branch only. Only statements in the branch of the statement at the top of the screen are viewed. This command can be very useful for printing. It can be somewhat disconcerting to jump to a statement with nothing under it with this viewspec set, and have everything but this statement disappear. 10A2

•

h -- Show all branches. This is the default, and undoes "g", "l", or "Q". 10A3

i -- User filter on. It is possible to write content patterns, or content programs, that cause only statements satisfying some kind of pattern to be displayed on the screen. If such a pattern or program is written and loaded, viewspec "i" says to view only statements passing the pattern. It is a command for experts only. If "i" is set, and there is no program loaded, it has no effect. 10A4

j -- Don't filter statements. This is the default, and undoes viewspec "i" or "k". 10A5

k -- Show next filtered statement. If a content pattern or program as described under viewspec "i" is in effect, viewspec "k" says to find the first statement passing the filter, and to show it at the top of the screen. From that point on, all statements are displayed, regardless of whether they pass the filter or not. 10A6

l -- Show plex only. Only statements in the plex of the statement at the top of the screen are shown. It is sometimes useful for printing. 10A7

u -- Recreate window after change. This is the default, and undoes the (very dangerous) viewspec "v". 10A8

v -- Defer recreating window. Beware!! This command is for experts only. Normally, when you delete a character, the other characters on the screen shift around to show the new orientation. With this viewspec on, the screen does not change, until specifically requested with a viewspec "f". It can save time if the system response is very slow, but should be used with extreme caution. AUGMENT knows what the screen

should look like after the edit, and your marks are interpreted relative to that screen, not relative to the screen you are viewing. To use this viewspec, it is best to edit backwards, since usually edits later in the file do not affect the screen positions of text earlier in the file (but not always!). If you get confused, set viewspec "f", to see where you are. If you must try out this viewspec, try it first on a file you don't mind destroying. Sometimes this viewspec gets set accidentally. If your edits do not seem to take effect, do a Show Viewspecs to see if you might have accidentally turned on viewspec "v". 10A9

A -- Level indenting on. This is the default, and undoes viewspec "B". 10A10

B -- Level indenting off. All statements, regardless of their level, are left-justified on the screen. It can let you view more of a file, and if the numbers are on, it is not too hard to deal with different levels. Note that the statements still have substatements, etc. Their structure simply is not being shown via indenting. 10A11

C -- Show statement names. This is the default, and undoes viewspec "D". 10A12

D -- Do not show statement names. The statement names and their delimiting characters (if any) are not shown on the screen. In this file, for example, if you turn on viewspec "D", the first words of most statements will disappear. See the section in this file <0188> on

•

statement names, if you do not know what they are. 10A13

F -- Recreate display. This viewspec is essentially identical to viewspec "f". In the case of "f", AUGMENT will only redraw if it thinks something has changed. Viewspec "F" says "Redraw the screen no matter what". Sometimes there is line noise, and the screen draws incorrectly, but AUGMENT thinks it is OK. In this case, viewspec "F" forces AUGMENT to redraw the screen. 10A14

K -- Show statement signatures. Every time a statement is edited, AUGMENT notes the time and date, and the ident of the person who did the last edit. You can see that information by turning on viewspec "K". Try it and see. Viewspec "L" turns it off. 10A15

L -- Do not show statement signatures. This is the default, and undoes viewspec "K". 10A16

O -- Turn on user sequence generator. Expert users may actually write AUGMENT code files, and change the order in which statements are presented on their screen. If you have written such a program and loaded it, this viewspec turns it on. 10A17

P -- Turn off user sequence generator. This is the default, and turns off viewspec "O". 10A18

Q -- Offsets indenting to level of statement at top of file window, when used with "l" or "g". If viewspec "l" or "g" is set along with viewspec "Q", the statement at the top of the file window is left justified, and all following statements are indented in the usual way relative to it. Try it -- it sometimes saves some space on the screen. It is turned off by "h". 10A19

Split screens 11

It is possible to split your AUGMENT screen into two or more windows, and

to view different files (or different parts of the same file) in the different windows. Each window on the screen has its own viewspecs, its own return ring, its own file return ring, and so on, associated with it.
11A

Any file window on your screen can be broken down into two smaller windows, using the command "Break Window". You can split the window vertically or horizontally, and the old window can be broken into unequally sized pieces. The contents of the present window will appear in one of the newly created windows; which one depends upon where you mark in the command. 11B

It's easier to do it than to explain it. Let us try. Suppose that you want to break the present window into two equal sized pieces, a left and right half, and to have the current information displayed in the left window. Start by jumping so that the next statement is at the top of your screen. 11C

Now, issue the command "Break Window Vertically (at) Center (of window) MARK (displaying in) MARK <OK>". For the final MARK, make sure that you mark to the left of the center. If you mark to the right, the information in the current window will appear in the right window. Go ahead and complete the command now. 11D

The command to delete a window is "Delete Window MARK (enlarging window at) MARK <OK>". The first MARK marks the window to be deleted and the second marks the window that is to expand into the newly available space. Only rectangular windows are allowed, so it is possible to mark certain illegal combinations of windows, and you will get an appropriate error message if you try something illegal. Go ahead now, and get rid of the new window that you just created. 11E

If you want to make windows of unequal size, instead of typing "c" for "Center", mark the position where you want the break to occur. If you wish a horizontal break rather than a vertical one, type "h" for "Horizontally" instead of "v" for "Vertically" after "Break Window". At this point, you might try creating some new windows of different sizes, both vertical and horizontal. Try breaking the broken windows to make more. Up to seven windows are allowed. 11F

Then, delete all windows but the one in which this file appears, and break this main window into two halves (a left and right half). The other window, instead of having any text displayed in it, should have the word "Empty" in it. To get a file in that window, issue a Jump (to) Link command (say to TUTOR2), and be sure to type the final <OK> when the cursor is in the empty window. Do so now. 11G

All of the Jump commands now have a slightly different meaning. The part that tells where to jump remains the same, but the jump will always take place in the window where the final <OK> is typed. The final <OK> has become more like a mark than an <OK>. Some of the mysteries of commands you learned earlier are now made clear. For example, in the "Jump (to) Origin" command, you were required to give a seemingly unnecessary <OK>. The reason is that if you have more than one window, you may have more than one file on your screen, and AUGMENT needs to know which file to jump to the origin of, and which window to present the origin of that file in. 11H

Do some experimentation now, and break windows, get different files in the different windows, set different viewspecs in the different windows, and try copying pieces of information from one file to another. Notice how easy it is to do this. From your point of view, the pieces of text may as well be in the same file. With broken windows, it is just as easy to edit

across files as it is to edit in one file, as you learned to do in the first tutor file. 11I

There are a number of problems that arise when windows are broken into smaller pieces. The maximum size of an AUGMENT statement is 2000 characters, which approximately fills one screen. If your windows are only half the full size, any statement with more than about 1000 characters cannot be entirely viewed in the half-size window. Without breaking the statement, or enlarging the window, there is no easy way to read the entire statement. This statement has that problem. Break it here: XXX. In fact, if the statement is the full 2000 characters long, there are times when the bottom line may not fit on the screen, due to line breaking, and you may have to resort to breaking the statement, even with a full-sized window. 11J

There is a command to enlarge a window against another window -- it is "Enlarge (window) MARK (to) MARK <OK>". The first mark tells which window

is to be enlarged, and the second tells the exact point to which it is to be enlarged. Try enlarging one of the windows on your screen. 11K

Using subsystems 12

After you learn about another subsystem by reading information you get through Jump (to) Locator or the online Help files, you can get to the by using the command "Goto <OPT> subsystem-name<OK>". When you are done, and want to return to Base, use the "Quit <OK>" command. More will be written on this topic later. 12A

Using the keyset 13

One piece of equipment at your AUGMENT 1200 work station has not been discussed. It is the five-key device to the left of your terminal. There is never a case where this device is required; if you learn to use it, however, you can speed up your editing by quite a bit. It is called a "keyset". 13A

If you are editing a document, you will find that in many cases, you will type two or three characters, then point with the mouse, and then type two or three more characters, followed by <OK>. If you are a touch typist, you probably have to glance down at the keyboard (and away from the screen) every time you need to shift your hand from the mouse to the keyboard. All of this costs time -- wouldn't it be nice to be able to keep one hand on the mouse, and still be able to type a few characters using your left hand only? Well, that is exactly why the keyset was designed. 13B

By typing certain "chords" on the keyset, you can indicate certain characters to AUGMENT. Typing an "a" on the keyset is exactly the same as typing an "a" on the keyboard. It does take some time to learn all the letter patterns, but it does not take too long to learn enough to save yourself time. 13C

Obviously, since you will only be typing with one hand, the keyset is slightly slower than a typewriter keyboard, but for short bursts of characters, you are saved the time it would take to reposition your hands from the mouse to the keyboard. Some experiments have indicated that the break-even point is about seven characters -- if you need to type more than about seven characters in a row, it is worthwhile to move your hand from the mouse to the keyboard. When you are inserting text, you will do 99 percent of your typing on the keyboard, but when you are editing, you will find that you use the keyset a lot. 13D

Welll, enough talk. Let us delete the extra "l" at the end of the first word in this statement using the keyset. Press the middle key, and let it go -- this is the code for "d" -- "Delete" should appear in the command window. Now, press the two rightmost keys down at the same time, and let them go. This will type a "c". Then, mark the "l", type a final <OK> on the mouse, and you are done! The mechanism is just that easy; you now are only faced with the minor (?) problem of learning the codes. 13E

The little card that has the viewspecs on one side has the keyset codes on the other. It looks rather complicated because, in fact, it is possible to type on the keyset (and mouse) any character that can be typed on the keyboard. With enough patience, you would not even need a keyboard at all!

13F

In the first column on the card is a series of combinations of the digits "0" and "1". "1" means to press the corresponding key on the keyset, and "0" means to leave it up. In the second column on the card is a list of the characters to which the codes correspond. Ignore the other columns for the moment. Check the codes for "d" and "c", and notice how they correspond to what you did to delete a character. Also notice that at the bottom of the column are a few other characters that can be typed. They include the most common punctuation marks, and space (all five keys pressed at once). 13G

Now, let us look at the other columns. In the second column, for example, is a list of the capital letters, together with a few more punctuation marks. These can be typed by holding down the middle mouse button, while typing the appropriate keyset combination. try replacing the lowercase "t" at the beginning of this sentence with a capital "T" by using the Replace Character command, typed from the keyset and mouse alone. First look up "r" for "Replace". It is the leftmost and fourth from the left keys typed at the same time. Type this now. Look up "c" for "Character", and type it on the keyset. Mark the "t", and then type a capital "T" by holding down the middle mouse button while typing a "T" keyset combination. Then a final <OK> (typed from the mouse) ends the command. 13H

You can read for yourself the other combinations available on the mouse and keyset. Note also that there are a few combinations that can be typed on the mouse alone which have not been mentioned until now. The two outermost buttons on the mouse, for example, (with no keys pressed) means <ESC>. 13I

Do not try to learn all the combinations at once -- in fact, do not worry about the capital letters, or the strange characters in the third column until you are quite comfortable with the basic lowercase letters. These are the combinations that will save you time. 13J

There are two basic approaches to learning the keyset -- the gentle, gradual approach, and the brute force approach. Brute force is much faster, but much more unpleasant. 13K

> The gentle approach: Learn just one or two letters at a time. A good pair to start with are "j" and "l", for "Jump (to) Link". This is probably the most common command, and every time you do a "Jump (to) Link", use the keyset instead of the keyboard. Then add a few more letters, like "d" and "c" for "Delete Character", or some other common letters, and continue gradually. Before long, you will realize that you know almost all of the letters, and you can do half an hour or so of the "brute force" approach to learn all the rest of the letters. 13K1
>
> The brute force approach: Put the keyboard out of reach, and set the

viewspec card in front of you and type absolutely everything on the keyset from the beginning, including logging into the computer. This will be very painful at first, but after a couple of hours, you will be amazed at how good you have become. 13K2

There is actually a method related to the brute force approach that some people have found useful. In any file (since you are going to do a command delete at the end anyway), type "Insert Statement (to follow statement at) MARK", and then begin typing the alphabet over and over

again, in order, using the keyset. Soon, your fingers will begin to learn the patterns, and you can "count up" to an uncommon character later. Thus, if you forget what the keyset combination for "k" is, but you remember the pattern for "j", just count up by one in the pattern that you have taught your fingers. 13K3

There are probably other methods that you can think up which would be best for you. It is a little extra work to learn to use this new tool, but it does pay off if you become proficient at using it. 13K4

There are quite a few patterns that can make the learning easier. Some of them will be pointed out here. 13L

The uppercase and lowercase letters have identical keyset combinations. The only difference is the second mouse button, which must be held down to type uppercase letters. 13L1

The control characters can all be typed with the same keyset combination as the corresponding letter, but using still a different mouse combination. 13L2

If you know anything about binary counting, notice that the letters go up in a binary progression. Hence, if you need to type a "k", and can't remember its combination, you can "count up" one from "j". 13L3

There is only a one-key difference between the digits and the corresponding punctuation mark that appears above them on the standard typewriter keyboard. 13L4

Two more hints, and you are on your own. The best scheme is to make sure that they are all required keyset keys and mouse buttons held down at the same time, then release the keyset keys, and a fraction of a second later, release the mouse buttons. If you press down an extra button, there is no way of undoing it. The typed character is the combination of all keys and buttons pressed between the time that the first one is pressed and the last one is released. This does provide a way out, though -- if you type too many keys or buttons, you can hold down all eight of them, and the combination will be ignored. Also, if you make a mistake, remember that the left mouse button by itself is a backspace. Good luck, and happy keysetting. 13M

Special characters 14

This section discusses all of the control characters used by AUGMENT. If you have an AUGMENT 1200 terminal, for example, you probably almost never need to type a control character, but if you are using another type of terminal, you may do it quite often. In fact, even with an AUGMENT 1200 terminal, you type special control characters all the time -- they just are not marked as such. For example, when you type <OK>, you are really typing <CTRL-D>. This section is meant to provide a short discussion of the uses of all the ASCII control characters. 14A

<CTRL-@> 14A1

This character is the ASCII NUL character (not to be confused with the AUGMENT <NULL> character), and should very seldom exist in any

file. It is often used internally by the computer to mark the ends of strings and other things. Many of the programs that deal with characters automatically throw away any characters of this type. You will have little or no use for it (as an AUGMENT user). 14A1A

<CTRL-A> = <BC> 14A2

This is the usual AUGMENT backspace character. 14A2A

<CTRL-B> = <RC> 14A3

This is the AUGMENT repeat command or <RC> character. If typed at the end of specifying a command, it will cause that command to be repeated over and over until a command delete is typed. 14A3A

<CTRL-C> 14A4

This is the Executive interrupt character. If you type it, any Executive program (including AUGMENT) will be interrupted, and you can then issue commands to the Executive. Typing it when AUGMENT is expecting a command is almost exactly the same thing as the Quit command. This character should NOT be typed while an AUGMENT command is executing (unless some error has occurred). It may leave a command half-finished, and a bad file may result. Typing "continue" at Executive will cause whatever program was executing to continue as if nothing had happened, so if you type it by accident, you can type "continue" immediately, and things will continue. You may need to press the LOCAL RESET key to get the screen properly refreshed. See <01275>. 14A4A

<CTRL-D> = <OK> 14A5

This is the same as the AUGMENT <OK> key. 14A5A

<CTRL-E> = <INS> 14A6

This is the same as the AUGMENT <INS> key, used to get into insert mode. See <0195>. 14A6A

<CTRL-F> 14A7

This key behaves much like <ESC> in file name and directory name recognition. The difference is that <ESC> attempts to fill out the entire file name (or directory name) and <CTRL-F> only tries to fill out the current section. In other words, suppose I have a file named ABCDEFGHIJKLMNOP.AUG and another named ABCDEFGHIJKLMNOP.TXT. If I want to delete the .TXT version, I would have to type all the way to the "T" before pressing <ESC>. Assuming I have no other files beginning with the characters "ABC", I could identify the correct file by typing "abc<CTRL-F>t<ESC>". 14A7A

<CTRL-G> 14A8

This is the ASCII BEL character, and it causes the bell on the terminal to ring or beep (assuming the terminal is so equipped). It

is sometimes useful for getting the attention of someone you have

linked to, in case that person is too engrossed in working to notice the "LINK from XXX ... " message. 14A8A

When you are printing using a workstation printer, and have asked AUGMENT to wait at the page breaks so that new pieces of paper can be inserted, <CTRL-G> restarts the printing at the beginning of each new page. Insert the paper, and then type <CTRL-G>. 14A8B

<CTRL-H> 14A9

This is an alternative backspace character, considered to be backspace by some Executive programs. You can usually use it as an AUGMENT backspace as well. 14A9A

<CTRL-I> = <TAB> 14A10

This is the <TAB> character. It can do different things, depending on your Profile Feature setting. A <TAB> typed at the "BASE C:" prompt causes the last search to be repeated. Inside a file, a <TAB> usually causes the text following it to be printed at the next tab stop. 14A10A

<CTRL-J> = <LF> 14A11

This is the line feed character. 14A11A

<CTRL-K> 14A12

In the Executive, this is known as a horizontal tab character, but it has no special purpose in AUGMENT. It is not a bad choice if you need a special character for some purpose, and you want to choose one that has no special interpretation by the system. 14A12A

<CTRL-L> 14A13

This is the form feed character. When it is sent to some printing devices, it causes them to advance to the top of the next page (top of form). It can be used to force a page break while printing, without going through the Output Processor. It is generally used by sequential file editors to mark the end of logical pages. 14A13A

<CTRL-M> = <RET> 14A14

This is the return character, <RET>. 14A14A

<CTRL-N> = <NULL> 14A15

If you have an AUGMENT 1200 terminal, this is the NULL key on your terminal. It is generally used to indicate "nothing". For example, if you are setting name delimiters, and you want to specify "no delimiter", simply type <NULL> (or <CTRL-N>). 14A15A

<CTRL-O> 14A16

This character is generally used to stop a process. If you are printing, and you decide that you do not want to wait for all the

text to be printed at your terminal, type this character. If you are running a commands branch, and it suddenly starts doing something wrong, this character will interrupt it. If you are compiling a program and want to abort the compilation, type <CTRL-O>. Because of various buffering schemes in your terminal and in the network, the key may not take effect immediately; often you must wait for some buffer or other to empty. 14A16A

<CTRL-P> 14A17

This character is not used by AUGMENT, and is often a good character to use if you need one that has no special interpretation. See <CTRL-K>. 14A17A

<CTRL-Q> = <HELP> 14A18

In AUGMENT, <CTRL-Q> is the <HELP> key. It enters the Help system, taking into account where you were in the command you were typing, and gives the best help it can on that subject. 14A18A

This character is also used (together with <CTRL-R> and <CTRL-S>) by TYMNET to stop terminal output. If, when you are typing your TYMNET user name, you type <CTRL-R>, you have armed the mechanism. The character can be typed before, after, or during the typing of your user name -- just type it before the final return character or semicolon. After this is done, <CTRL-S> can be typed at any time, and TYMNET will essentially freeze. No characters that you type will go to the computer, and no computer characters will come to you until <CTRL-Q> is typed. As soon as this happens, the characters are released in both directions. This is useful if you are printing out a long document that scrolls on your screen. Type <CTRL-S> to stop it, read it, and then type <CTRL-Q> to continue the scrolling. If you need to type a real <CTRL-S> to the Executive host for some reason, simply type <CTRL-S><CTRL-S><CTRL-Q>. The first character freezes the network, the second is saved, and the first is released when the final <CTRL-Q> is typed. 14A18B

<CTRL-R> 14A19

This character is used by AUGMENT to retype the statement so far. It is like <CTRL-S> in Mail. It is also used by TYMNET (see <CTRL-Q> above). 14A19A

<CTRL-S> 14A20

This character is not used by AUGMENT, but it is used (together with <CTRL-Q> and <CTRL-R>) by TYMNET. See the discussion of <CTRL-Q>, above. 14A20A

In the Executive Mail program, <CTRL-S> is used to retype the entire text so far. This is convenient, since after a certain number of errors are typed and backspaced over, the message may become unreadable. <CTRL-S> simply asks the program to list out a clean copy of what has been typed so far. 14A20B

<CTRL-T> 14A21

This character asks for a load average and an indication of what is happening. If you are not sure whether anything is happening, you can type this character. If nothing happens, you are not talking to

the computer. If you are connected to the computer, something should happen. If the response is "Running ...", the computer is doing something, and not waiting for you. "Waiting ... " generally means that it is waiting for you to type something, although not always. If you are running Telecopy, for example, it may be waiting for the remote computer to do something. You can tell for sure if it is waiting for you by looking at the number followed by a letter. If the letter is "F" (for "front end"), it is waiting for you. 14A21A

For example, "Running 177425B" means it is running, "Waiting 46625F" means it is waiting for you, and "Waiting 177232R" means it is waiting for a remote process. 14A21B

The time of day, your connect time, and your CPU time are also displayed. 14A21C

<CTRL-U> = <OPT> 14A22

This is the AUGMENT <OPT> character. 14A22A

<CTRL-V> = <LIT> 14A23

This is the <LIT> character. Using it, you can get most unusual characters into an AUGMENT file. For example, if you are typing a statement that is part of a Process branch, you will probably need to end it with an <OK> (=<CTRL-D>). If you just type the <OK>, that will end the Insert command without actually inserting the <OK> character. To get an <OK> into a file, simply type <LIT> first, then the <OK>. <LIT> essentially means: "Let the next character, whatever it is, go by, and do not try to interpret it in the usual way". To get a <LIT> into a file, simply type two of them in a row. 14A23A

There are a few characters that cannot be inserted in this way. <CTRL-C> is intercepted and acted upon by the Executive, for example. <CTRL-^> is intercepted by the terminal. If you are connected through Telnet, a <CTRL-Y> would be intercepted by that program. Basically, the only characters that can be inserted in this way (which includes most of them) are characters that actually make it through to AUGMENT. 14A23B

<CTRL-W> = <BW> 14A24

This is the AUGMENT backspace word or <BW> character. 14A24A

<CTRL-X> = <CD> 14A25

This is the AUGMENT command delete or <CD> character. If you are typing any command, and have not yet given the final <OK>, this command will throw the entire command away and let you issue another one. 14A25A

<CTRL-Y> 14A26

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 51

This character is used in the Conference subsystem as an escape character. Any member of a conference can exit by typing this character, even if he or she is not holding the gavel. If the person who initiated the conference types a <CTRL-Y>, the whole conference is dissolved. 14A26A

The same character is also used as the default Telnet interrupt

character. If you are connected to another host via the ARPANET, and you find it necessary to talk to the Telnet program, type <CTRL-Y> and enter a single command to Telnet. After you have ended the command, you will be back where you were. It is possible to change this to another character if for some reason it is important to type <CTRL-Y>s to the other host. 14A26B

<CTRL-Z> 14A27

This character is used in AUGMENT to get you out of any subsystem and back to Base. In Executive, it is the end-of-file character, and is used to end messages sent via Mail or Hermes. 14A27A

<CTRL-[> = <ESC> 14A28

This is the escape or <ESC> character. In AUGMENT, all this character is used for is file name recognition. A partially typed file name will be filled out (if possible) when <ESC> is typed. If more than one file matches the characters typed so far, nothing happens. If no files are recognized, a "$" is echoed. <ESC> works slightly differently in AUGMENT than in the Executive. In AUGMENT, a file name with extension ".AUG" will always be selected over a file name having a different extension. 14A28A

<CTRL-\> 14A29

This is the SCREEN SAVE on your AUGMENT 1200 terminal. It is usually intercepted by the terminal and rarely makes it into an AUGMENT file. 14A29A

<CTRL-]> 14A30

This is the GRAPHICS key on your AUGMENT 1200 terminal. It is usually intercepted by the terminal and rarely makes it into an AUGMENT file. 14A30A

<CTRL-^> 14A31

This is what the SCREEN SWITCH key on your terminal sends. The character is usually intercepted by the terminal and does not make it into AUGMENT files. 14A31A

<CTRL-_> 14A32

On the ARPANET, this character can be used to hold the output. Typing any character will undo this effect (in fact, the same key can be used to both hold and release output). 14A32A

Sequential files 15

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 52

Introduction 15A

AUGMENT is able to do many of the things that it does because its files are not organized in the same way as most of the files on the computer are. If you only used AUGMENT, you could probably avoid dealing with non-AUGMENT files most of the time. Unfortunately, if you want to interface with users of other systems, especially different mail systems, you will need to learn a little bit about how to deal with non-AUGMENT files. (Even as an AUGMENT user, you actually make use of non-AUGMENT files from time to time, but their use is somewhat hidden.) 15A1

Although there are a number of different file types, you will probably need to deal only with a few of them. The most important types from an AUGMENT user's point of view are AUGMENT files and sequential files. Both types are used for storing text. There are other files which contain compiled programs and other information, but you will probably not need to deal much with these. 15A2

File extensions 15B

In the vast majority of cases, a file's extension tells you its type. For AUGMENT files, the standard extension is "AUG", but for some older files the extension may be "NLS". There is no reason that an AUGMENT file must have an "AUG" extension or that a non-AUGMENT file must have a some other extension. It is just a convention that you must go out of your way a little bit to break. 15B1

Below is a list of some of the more common extensions, and a short description of their usual meaning. The list is not intended to be complete, but it should take care of most of the usual cases. 15B2

AUG 15B2A

The standard extension for AUGMENT files. 15B2A1

NLS 15B2B

An old extension for AUGMENT files. AUGMENT will recognize these files as well as files with the AUG extension. 15B2B1

PC 15B2C

Files having this extension are modifications files (and are what the Delete Modifications command deletes). 15B2C1

WPF 15B2D

This is just a standard sequential file, but is usually created by the AUGMENT Print command. The letters stand for "workstation printer file". When the workstation printer is running, any files with this extension are copied out onto the printer, and the file is then deleted. To print any sequential file, in principle, all you have to do is change the extension to "WPF" while the workstation printer is running. Of course, it will also be automatically deleted after printing. 15B2D1

HPF 15B2E

This stands for "held print file". It has exactly the same contents as a "WPF" file, but will not automatically be printed when the workstation printer is running. Files with this extension are the usual method by which print files can be saved. 15B2E1

{IDENT} 15B2F

The notation above stands for an extension that is the same as your ident. 15B2F1

CA, SK, GR, BE 15B2G

These are extensions associated with various AUGMENT programs.

"CA" stands for a compiled content analyzer program, "SK" for a sort key, "GR" for a grammar, and "BE" for a back end. This list is not all inclusive. 15B2G1

TXT 15B2H

This is the standard extension for sequential text files. Your non-AUGMENT mail is delivered to a file named "MESSAGE.TXT". 15B2H1

SAV 15B2I

This extension means that the file is a program ready to run. If you look at the files in the SUBSYS directory, for example, you will find that many of the files have a SAV extension. Some examples are "AUG.SAV", "HERMES.SAV", and "AUGMENT.SAV". This extension means that the name can be typed directly at the Executive "@" sign, and the appropriate program will be run. 15B2I1

others 15B2J

Various programming languages have their own conventions. For example, if you write programs in BASIC, the files you generate will have the extension "BAS"; in PASCAL, they will be "PAS", and so on. Relocatable compiled or assembled files will have the extension "REL". Some files have no extension at all; their names might look something like "NAME.;3". There is no reason that you cannot make up your own extensions having your own meaning. 15B2J1

Creating sequential files 15C

The basic command to create a sequential file from an AUGMENT file is "Create Sequential". This will create a sequential file formatted exactly the same way as the information is presented on your display screen. If numbers are on, numbers will be on in the sequential file, if blank lines are on between statements, there will be blank lines in the sequential file, and so on. The size of the sequential file created can be controlled by "g" and "l" viewspecs. 15C1

When Create Sequential is used, the statements are broken at appropriate word breaks, and return characters appear in the sequential file

produced. There is another command similar to "Create Sequential" called "Create Assembler", which is very much the same, except that it does no line breaking except at the ends of statements, where a return character is inserted. Another difference is that Create Assembler does not indicate indenting with three spaces -- it uses a <TAB> character. Statements indented two levels are printed with two leading tabs, and so on. Various assemblers prefer to see the output in this form, and it can be useful for other things as well. 15C2

Conversion to AUGMENT files 15D

Converting AUGMENT files to sequential files is quite straightforward; the opposite problem can be quite difficult. There are two reasons for this. The first is that there must be some way of telling AUGMENT where one statement ends and the next one begins. The second is how to tell AUGMENT what the level indenting should be. 15D1

If the file to be converted was originally constructed with the intention of eventually converting it to an AUGMENT file, these will not be problems, but if it was not, there may be some problems. Indenting

done in the original file may be interpreted as level changes in the converted file where they should not be, the spacing may not have been done in a consistent manner, and there are other problems. In many cases, the easiest thing to do is rather than get the conversion perfect the first time, copy it in as well as possible, and then use the AUGMENT editor to correct any mistakes that the automatic process may have made. 15D2

The basic conversion command for sequential-to-AUGMENT conversion is the "Create Augment" command. It will ask for the name of a sequential file to be converted, and for a place in an AUGMENT file to put the converted text. If you want to convert a sequential file into a new AUGMENT file, you must first create the new AUGMENT file with the Create File command, and then use the Create Augment command to do the conversion. You then have the choice of whether to break statements after one or two return characters in the sequential file, or you have the third option of Assembler. 15D3

Breaking after one return character makes each line of the sequential file a new AUGMENT statement; breaking after two looks for two return characters in a row (i.e. a blank line) before ending one AUGMENT statement and beginning another. The third option, Assembler, will break statements after each return character, but will convert the paragraphs into statements all of the same level. No attempt to impose AUGMENT hierarchical structure will be made. 15D4

Thus if you are creating a sequential file with the intention of eventually converting it to AUGMENT, the best way is to insert two return characters at the end of each paragraph (statement). 15D5

AUGMENT will try to guess how to do the level adjustments based on the indenting in the sequential file. If each level in the sequential file is strictly indented three spaces compared to the last one, this conversion technique will work well. If the sequential file was created without AUGMENT in mind, there are likely to be many places where the relative indenting does not come out right. 15D6

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 55

If the paragraphs in the sequential file were typed with no blank lines between them, sometimes it is easier to create the AUGMENT structure using only one return character between the statements, and then use the Append All command to append all the statements in the groups forming the actual paragraphs. This technique can take a long time if the file is large. 15D7

If you have control of the creation of the sequential file that is planned to be converted eventually for AUGMENT, make it look like an AUGMENT file printed with viewspecs "y". Then use the command "Create Augment ... (using) Two (<RET>s to end statement)". 15D8

The following is a special trick that is often quite useful. Suppose you have an AUGMENT file that does have structure, and for some reason you want to get rid of the structure. This can be important sometimes for sorting purposes, since the Sort command only sorts statements at the same level. You can simply create a sequential file from the AUGMENT file, but with viewspec "B" on (no level indenting). Then use the Create Augment command to recopy the new sequential file back into another AUGMENT file. The sequential file will have no leading spaces, so the new AUGMENT structure will have nothing but top-level statements. 15D9

What a job is 16A

When you log into Executive, you have created an Executive job. Almost all of the time, you will be running a single job, although in some cases it is possible to run more than one. Running only one job is not much of a restriction -- it is possible to do many things at once in a single job. This section describes how to control your job -- how to run more than one program, how to switch your job's attention to a different program, and so on. 16A1

Detached jobs 16B

Normally, a job is attached to a terminal. In other words, the characters typed on the terminal go to the job, and some of the output from the job goes back to the terminal (some things may also be written into files, etc.). Certain types of jobs do not require any user interaction, and there is really no reason that they need to be connected to any terminal to run. All of the input and output go to files, and it is silly to waste a terminal connection to run those jobs. 16B1

Jobs running without benefit of an attached terminal are called "detached jobs". Sometimes jobs get detached accidentally, too. If your network connection breaks down, or if you pull the plug on your terminal, your job will become detached. When this happens, the processing of your job stops, but the job is not logged out. 16B2

It is possible to reattach to your detached job and continue as if nothing had happened. If you do not reattach to your job, it will be automatically logged out after 15 or 20 minutes. A normal job will also be automatically logged out after that long if no characters are typed to it. If you have refused autologout, your detached job will stay around forever or until the system comes down, whichever comes first.

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 56

16B3

You can intentionally detach a job by issuing the Executive command "detach". This is sometimes a good thing to do if you really need a different job (you need to log in as someone else, for example), if you want to log into a different computer, or if you want to run the same job on a different terminal. The reason that you would detach rather than log out is that for some reason, you would like to come back to exactly where you were before you detached. 16B4

If your job has become detached (either accidentally or intentionally), you can use the Executive "attach" command to reconnect to it. Suppose your user name is "oedipus" and your password is "rex". Type the following at the Executive "@" sign to reattach: 16B5

```
@attach oedipus rex<RET>
```
16B5A

The password will not be echoed, as usual. You will then be asked for your Executive job number, and you can just press <RET> if there is only one such job. The case where there is more than one detached job is discussed later. 16B6

If you detached intentionally, you will know the exact state of your job, and you know how to continue. If your job was detached accidentally, you may not know exactly what its condition is. In particular, if it was an AUGMENT job, when you reattach, AUGMENT continues as if nothing had happened; it has no idea anything happened -- no one typed it any characters for a while is all. For this reason, AUGMENT sees no particular reason to redraw the screen, so you should

press the LOCAL RESET key on your terminal or set viewspec "F" to force a screen recreation. 16B7

Occasionally, another problem arises -- the job does not think it is detached. This happens more frequently on TYMNET than ARPANET. If this happens, you can still reattach, but the procedure is a little different. (In fact, it is possible to rip a running job away from another terminal using this method.) 16B8

First, you need to find out the Executive job number of the job. This can be done by typing: 16B9

@where oedipus<RET> 16B9A

The Executive will respond with something like: 16B10

TTY66, Job 15, AUGMNT 16B10A

To attach to such a job, type: 16B11

attach oedipus rex 15<RET> 16B11A

If you have more than one detached job, you can use exactly the same technique to find out the job numbers, and reattach. 16B12

When you attach to a job that is not detached, you must confirm with an extra <RET>, and a <CTRL-C> is automatically sent to the job. If you want to continue, type "continue" in the Executive. 16B13

Interrupting with <CTRL-C> 16C

At any time during a session, you have a certain context in which you are working. You may be in AUGMENT, you may have certain subsystems loaded, and you may be at a certain statement within a certain file. All of this information is part of your context. Your context also includes any command that happens to be executing at the time. 16C1

At any point, you can interrupt all of this by typing a <CTRL-C>. This halts any process that happens to be running, and returns you to the Executive. (Later we will see that more than one Executive may be present -- in this case, <CTRL-C> returns you to the Executive running the current program.) 16C2

This interruption does not alter your context at all -- it simply gives you the opportunity to type commands to the controlling Executive. A "continue" command typed in the Executive simply restarts everything as if nothing had happened (unless, of course, you altered your context while in the Executive). Basically, however, if you type a <CTRL-C>, followed immediately by a "continue", there will be no net effect. 16C3

There is a certain set of Executive commands that do not alter your context. Any number of them can be run, and followed by a "continue", and you can go on as if nothing had happened. Listed below are some of the more common Executive commands, together with a discussion of how each of them affects your context. 16C4

Account -- changes your account; does not affect context 16C4A

Advise -- does not affect context 16C4B

Append -- does not affect context 16C4C

Archive -- does not affect context 16C4D

Assign -- does not affect context 16C4E

Attach -- does not affect context; your current job will be detached, however 16C4F

Break -- does not affect context 16C4G

Bye -- does not affect context 16C4H

Cd -- except for changing your connected directory, does not affect context 16C4I

Change -- does not affect context 16C4J

Connect -- except for changing your connected directory, does not affect context 16C4K

Continue -- does not affect context 16C4L

Copy -- does not affect context 16C4M

Daytime -- does not affect context 16C4N

Ddt -- does affect context; loads DDT, and starts running it. If you know DDT, you can sometimes get back your context. 16C4O

Deassign -- does not affect context 16C4P

Delete -- does not affect context 16C4Q

Detach -- does not affect context 16C4R

Directory -- does not affect context 16C4S

Downtime -- does not affect context 16C4T

Dskstat -- does not affect context 16C4U

Edit -- completely destroys context, and loads the TECO editor 16C4V

Exec -- does not affect context, but creates a new Executive; Pop and Continue to get back context 16C4W

Expunge -- does not affect context 16C4X

Filstat -- does not affect context 16C4Y

Formfeed -- does not affect context 16C4Z

Fullduplex -- does not affect context 16C4AA

Get -- completely destroys context 16C4AB

Group -- does not affect context 16C4AC

Halfduplex -- does not affect context 16C4AD

As was stated earlier, it is possible to do more than one thing at a time in a single job. An Executive job is made up of a number of forks, each one of which can run a program. Any fork can, in

principle, run subforks. If your job has ten forks, it could simultaneously be running ten programs. 16D1A

Most user jobs have a fairly simple structure: The Executive runs in one fork, and the user program runs in another fork controlled by the Executive fork. The Executive is the program that interprets the characters you type at the Executive "@" sign. AUGMENT is slightly more complicated -- it is actually made up of two forks, the front end and the back end, and the Executive runs the front end, which in turn controls the back end fork. 16D1B

When you use the "Goto Exec" command in AUGMENT, you start up another fork running the Executive program, and running under the control of the back end. You can Goto Exec (making a new instance of the Executive program), and run a program under its control; when that program is finished, you can quit back to AUGMENT, exactly where you

left it. It is possible to start up new forks in other ways, which will be discussed later. 16D1C

First, some warnings should be stated concerning the use of multiple forks in a job. When programs are run, they often open files in such a way that the files are not openable by other processes. If you have two forks that both need to modify the same file, errors may occur. The main danger here comes from trying to run AUGMENT in two different forks. A typical mistake made by users is that they will be running AUGMENT, Goto Exec and run some other program, and finally, enter AUGMENT again, not by quitting back to the original version, but by starting up a new version. The Executives look exactly the same, and it is easy to forget that you are not running in the main one, but rather in an "inferior Exec". Typical symptoms for this problem are eror messages like "PC open fail ...", or "can't open file blap". It probably means that another fork (or job) has the necessary file open. 16D1D

If you are planning to run different programs in the different forks (AUGMENT and Hermes, AUGMENT and Mail, Hermes and Startrek, etc.) there will be no problems. Hermes and Mail would certainly conflict, since they both need to write in the same message file. 16D1E

Using multiple forks 16D2

As was stated before, the Goto Exec command in AUGMENT will fire up a new Executive fork (an inferior Exec). After you are finished with a new fork, type "pop" or "quit" to the Executive, and you will be returned to the controlling fork. If that was AUGMENT, you can continue there as if nothing had happened. 16D2A

Another way to make a new fork is directly from the Executive with the command "push" or "exec". The context of the superior fork is not damaged, and new programs can be run in the inferior fork. For example, suppose you are running AUGMENT, and you suddenly have a desperate need to play Adventure. You could proceed as follows: 16D2B

Type <CTRL-C> 16D2B1

Type "push" 16D2B2

Type "adventure" 16D2B3

When you are finished playing Adventure, exit that program, and

type "pop", then "continue". The "pop" returns you to the main Executive (destroying the contents of the inferior Exec), and the "continue" continues running the program in the main Executive. 16D2B4

There is no particular reason to prefer this method to using "Goto Exec" -- it is just another way to proceed. 16D2C

There is a third (sometimes very handy) way to run a new job, and that is as an "ephemeral" (see <01319>). 16D2D

Executive command types 16E

•

DI, 10-Sep-96 16:49-PDT < USER:DIANA, TUTOR4.AUG.41, > 61

Simple commands 16E1

If you type many of the Executive commands (for example, to examine your directory, to delete a file, or to link to another user), your context will not be altered. Thus, if you just need to issue a simple command like this, type <CTRL-C>, issue the command, and then type "continue" to be back where you were. Most of the commands that you can find out about by typing "?" at the "@" sign are like this. 16E1A

Normal programs 16E2

Every copy of the Executive is capable of running one program at a time. If you are running AUGMENT, Quit and Continue, you will be back in AUGMENT. If, however, you Quit and type "hermes", any context of your AUGMENT job will be lost. Hermes will be the current program. Most programs are like this. 16E2A

Ephemerals 16E3

A certain class of programs are called "ephemerals". These are big enough that they actually require a fork of their own to run in, but when they are run, a new fork is automatically created for them, and as soon as they end, the fork is eliminated. It is impossible to save the context of these programs in the usual way. The Executive command "grpsts" is of this sort. If you are running AUGMENT, Quit, and type "grpsts", when Grpsts ends, a Continue will put you back into AUGMENT. 16E3A

Any program can be run like an ephemeral. Just type "erun programname" at the Executive. This automatically takes care of the fork creation and deletion for you, but makes context saving more difficult. 16E3B

Batch jobs 16F

Use the Jump (to) Locator command to find the online documentation about this subject. 16F1

•