1 Welcome to the Master Level!

Get ready to plug in to the best features of BEX! Master Level BEX is only available when your Apple has at least 128K memory. All Apple IIc's and IIgs's qualify; no Apple II Plus can. If you have an Apple IIe without an extended 80 column card, you will have to install an extended 80 column card to make use of the Master Level. Appendix 4 discusses sneaky ways to take advantage of some Master Level features when your Apple has only 64K memory.

What to Expect from this Manual

To use the Master Level, we assume that you have read, understood, and practiced the material in the User Level. We expect that you are familiar with chapter selection, editing, printing, and Replace characters. The User Level summarized much of the material that you learned at the Learner Level. But at the Master Level, we focus solely on the new features. A summary of all the features available at the Master Level is in the booklet we call the Thick Reference Card.

Part 1: Overview of Master Level Features

BEX's prompts are much shorter at the Master Level. Instead of prompting `Main Menu:` BEX prompts `Main:` Whenever BEX wants a chapter name or drive number, the prompt is simply `Chapter:` If that's not brief enough for you, read Section 9 to learn how to change BEX's prompts yourself. And if you're feeling truly adventurous, Section 9 also explains how to change the braille translation tables.

Speeding Up BEX

Section 2 introduces the *Ready* chapter. Up to now, the chapters you have worked have always been stored on disk. The Ready chapter is stored in the Apple's memory. You can copy its contents to disk when you want to. For the Apple IIe and IIc, the Ready chapter can be six BEX pages; for the Apple

IIgs, the Ready chapter is 20 BEX pages. Because moving between pages in the Ready chapter happens totally in memory, it's a *lot* faster.

Section 3 explores extended disk systems. In addition to two 5.25-inch disk drives, at the Master Level you can use 3.5-inch disk drives, the Sider hard disk system, and RAM drives. When you load your BEX program disk on to a RAM drive, BEX really flies!

Greater Input/Output Control

At the Master Level, you can turn any of BEX's four output channels off and on when you wish. You can also change screen size at the Menus as well as in the Editor. You can have a braille or print device capture a small part of the computer dialogue for future reference.

You can set up *automatic procedure chapters* that record your keystrokes as you use BEX. At a later point you can replay these auto chapters, similar to running a player piano.

As you configured at the Learner and User Level, you probably wondered what a class S - Specific printer was. Well, Master Level Section 5 tells all. There's a new group of format commands that send escape codes to a specific printer. Section 5 also explains how to embed format commands within words; how to insert discretionary hyphens in your text, and even how to write about BEX format commands.

Contextual Replace

The largest Section in the whole of BEX is Master Level Section 6. That's because it provides the ins and outs of Contextual Replace. You can turn replacing on and off within a chapter. You can specify whole classes of characters in one stroke. You can ask for much more restrictive conditions than are possible with regular Replace characters. Without a doubt, Contextual Replace is the most intricate and powerful part of BEX. Contexual Replace made it possible for us to produce this manual in both large print and braille from one set of data.

Part 2: Advancing to the Master Level

The way to tell BEX to unlock all these new features is by creating a Master Level configuration. To set up a Master Level configuration, answer the `Enter configuration:` prompt with the `&` ampersand character. The configuration questions at the Master Level are identical to the User Level questions with one exception: the final question is `Do you have an extended disk system?` You definitely want to answer Y when you have 3.5-inch drives or an extra memory card in your Apple. See Section 3 for the details.

Enjoy!

2 The Ready Chapter

The *Ready* chapter is a special chapter which you can create and use only at the Master Level. We call it the *Ready* chapter because it is always ready to use, and because you specify it using the right bracket, which the Echo speaks as "ready." The Ready chapter does not exist on any disk drive; instead, it exists in the auxiliary memory created by your extended 80-column card. The size of the Ready chapter depends on your Apple model. BEX keeps up to six Ready chapter BEX pages in the auxiliary memory with Apple IIc and the Apple IIe; with the Apple IIgs, the Ready chapter is 20 pages long.

You create and manipulate the Ready chapter much as you would any other BEX chapter. Because manipulations on the Ready chapter do not involve floppy disk access, they happen in a flash. Moving from one Ready chapter page to another, for example, is just about instantaneous. Using the Ready chapter minimizes disk swapping when you have a one-drive system.

- ***Gone in a Flash***

The Ready chapter also disappears in a flash when you turn off computer power. However, it remains intact when you do a *warm boot;* that is, when you reboot without interrupting power. You can do a warm boot by depressing control and open-Apple, then pressing the Reset key; or by

entering `PR#6` at the BASIC prompt. Because a warm boot does not change the Ready chapter, you can reboot with a different configuration or establish a new configuration without erasing the Ready chapter. However, we recommend you save the Ready chapter to disk, if possible, before doing a warm boot, in case of loss.

The chapter name for the Ready chapter is just one keystroke: `]` known as the right bracket. To use the Ready chapter with any BEX function, enter its single right bracket name at any `Chapter:` prompt. The crucial difference between the Ready chapter and a chapter on disk is permanence. When you want a lasting version of the information in the Ready chapter, you must copy it to a disk chapter with a different name.

When you turn on power and boot BEX or when you kill the Ready chapter, it has no pages. Until you turn off power (or your neighbor blows a fuse) BEX keeps track of it. Because you can lose the Ready chapter easier than a chapter on disk, we recommend you back up the chapter often.

Part 1: Uses for the Ready Chapter

There are as many uses for the Ready chapter as there are for any BEX chapter. When you are doing a lot of work on one chapter, it is very efficient to copy it into the Ready chapter and proceed to edit and operate on the Ready chapter. The Master Level requires the additional 64K memory from an extended 80-column card, so in addition to the Ready chapter, the routines for printing, the Editor and Replace characters are always loaded in RAM. These operations and some others, like Copy chapters, do not require disk access. When you use the Ready chapter, BEX glides along with barely a disk access.

The main hurdle in using the Ready chapter is remembering that it is there. Now that you are at the Master Level, you are encouraged to change your habits and use the Ready chapter.

Whenever BEX prompts for `Chapter:` you can use the Ready chapter: in Print chapters or Multi-function print, as the source or target for braille translation or Replace, in Input through slot, or even as a transformation

chapter or automatic procedure chapter. If you have a one-drive system, you can use it as the target chapter when reading a ProDOS textfile into BEX.

Suppose you want to produce a letter in both print and braille. You write it in print form in the Ready chapter. You then copy it to a chapter called `ULTIMATUM` on disk. Then you use the Ready chapter as both source and target chapter for grade 2 translation and replacing with a transformation chapter. You copy the Ready chapter, now containing your final braille, to the `ULTIMATUM2` chapter.

The Ready chapter is perfect for temporary information. For example, to address an envelope, you can write the address material in the Ready chapter and then print the Ready chapter. You never have to save it to disk.

- ***Editing the Ready Chapter***

Editing with the Ready chapter is a treat. Moving between pages, and in and out of the Editor do not entail disk access time.

When you enter `]` at the Editor `Chapter:` prompt, you create a fresh Ready chapter if it is not already occupied. To use the Ready chapter for editing something already on disk, copy the disk chapter to the Ready chapter.

When you edit in the Ready chapter, you still have a regular-size clipboard of up to 4096 characters. As in any other chapter, the editing work on your current page is only in the page buffer until you save the page in memory. The current page is saved in memory when you change pages, enter control-P 0, or press control-Q.

When you leave the Editor with control-Q, your current Ready page is saved, but not on disk. When you want to save it to disk, use Copy chapters with `]` as the source chapter.

You may have more pages in your Ready chapter than your computer's Ready chapter page limit. BEX automatically saves extra pages to disk. We discuss this further in Part 2: Ready Chapter File Structure below. Of course, you are making most efficient use of the Ready chapter when you keep to the

page number limit, which is 20 for the Apple IIgs, and six for the Apple IIc or IIe.

When you want to clear the Ready chapter, so that it contains no text, you can use Kill chapters on the Second Menu. Kill chapters deletes any pages in the Ready chapter which happen to be saved on disk.

- ***You Can't Change the Ready Chapter's Name***

There is, however, one operation which does *not* work with the Ready chapter. This is option N - Name change for chapters on the Second Menu. You can't specify] as either the source or target chapter. The Ready chapter must be called] and only the Ready chapter can. Therefore, you can't specify] as either source or target for option N - Name change for chapters on the Second Menu.

Part 2: Ready Chapter File Structure

The file structure for the Ready chapter resembles the ordinary chapter file structure. Its *page files* are lettered sequentially:].A then].B and so on. The *files*].A through].F (or].A through].T on the Apple IIgs) are stored in auxiliary memory instead of on disk. Each of these files is actually a pocket of auxiliary memory which can hold up to 4096 characters. The directory for the Ready chapter is also kept in memory. Files].G (or].U on the Apple IIgs) and beyond, are disk files saved on your default data drive.

As with ordinary BEX chapters, the correspondence between page numbers and the one-letter file extensions can vary. When cutting pages and Page Menu manipulations are not part of a chapter's history, the page file extensions for pages 1, 2, and so on are in alphabetical order. As you may recall, page manipulations change the correspondence between the page number and its letter extension. When you kill pages or move them in the Edito, the new pages keep their old extensions. When you cut a page or use Grab pages, each newly created page file takes the next available extension

for that chapter. When you copy a chapter, the copy uses the same extensions as the original.

These considerations are equally true for ordinary BEX chapters and for the Ready chapter. They are just more crucial for the Ready chapter, since you want to use extensions `.A` through `.F` or `.A` through `.T` wherever possible.

Suppose you have a seven page chapter called `MANIFESTO` whose page file extensions are all in alphabetical order. You see its pages listed as `MANIFESTO.A` through `MANIFESTO.G` when you do a disk catalog. You then kill pages 3 and 4. When you do a file list, here is what you get:
MANIFESTO.A
MANIFESTO.B
MANIFESTO.E
MANIFESTO.F
MANIFESTO.G
MANIFESTO

Next, you copy chapter MANIFESTO to the Ready chapter. Page 5, with its *G* letter extension, is written to disk, even though you are not over the page limit. When you Copy chapters, the letter extensions remain the same. Any page with extension `.G` is *always* written to disk, no matter what page number it is in your chapter. If for some reason the first page of your Ready chapter has extension *Go,* that page is written on disk, and requires disk access whenever you move onto it.

Three options on the Second and Page Menus solve this problem. Options M - Merge chapters and A - Adjust page sizes on the Second Menu, and option G - Grab pages from another chapter on the Page Menu create new chapter directories. These options use alphabetical order for extensions as they create new page files.

For example, suppose you copy the chapter MANIFESTO using `]` as your target chapter. When you do a file list of `]` on the Page Menu, you see that page 1 has the letter extension *Go,* and is written to disk. Next, you use Merge chapters, specifying just MANIFESTO for the source chapter, and the Ready chapter for the target chapter. When you do a file list of the Ready chapter this time, you see that the page letter extensions are in alphabetical order. When you use Merge chapters instead of Copy chapters to move the chapter MANIFESTO into the Ready chapter, you solve your disk access

problem. No pages are written to disk. This also happens with option A - Adjust page sizes on the Second Menu, and option G - Grab pages from another chapter on the Page Menu.

Page manipulations on the Ready chapter itself, as well as some Editor commands such as control-C control-P to cut pages, also take its page extensions out of alphabetical order. When you have unnecessary disk files, you may wish to straighten the page files. For example, you can use Merge chapters with chapter `]` together with no chapter, or Grab pages from chapter `]` into a disk chapter, and then copy that disk chapter back into the Ready chapter. In the process, you reletter your pages alphabetically, and you also back up the Ready chapter on disk.

- ***Fix Chapters and the Ready Chapter***

When you use Fix chapters on the Ready chapter, the result is different from Fix chapter with a disk chapter. Because there is no disk file for the file extensions *A* through *Go,* or *A* through *T* on the Apple IIgs, BEX can't know how many characters are in each page. Therefore, every Ready chapter page in memory is restored to 4095 characters. It is as if you did `RUN 999` to all the pages in memory. When you need to use Fix chapters on the Ready chapter, you must delete excess material in the first six pages on the IIc or IIe, or the first 20 pages on the IIgs. However, Fix chapters works normally with any Ready chapter page written to disk.

- ***Keep a Disk in the Default Data Drive***

One of our staff was editing a six page Ready chapter without a disk in her default drive. When she cut a page, she created a seventh disk page, which needed to be written to disk. Since there was no disk in the drive, BEX could not save the page. She got the message `Cannot write to disk. Insert a data disk in drive 1 and press any key.` This created a one page chapter called `SAVE` on drive 1. Instead of saving the page as a part of her Ready chapter, it was saved into a separate chapter.

After getting over her panic, she realized there was a way to integrate her data. First, she copied chapter SAVE to a data disk in her default drive. Then

she pressed S to move to the Second Menu. There, she used option M - Merge chapters, with source chapters] and SAVE, with SALVATION as the target chapter. Now she had the seventh page along with the rest of the text from her Ready chapter. With that done, she deleted the SAVE chapter. As her last step, she copied the text in SALVATION back into her Ready chapter, only this time she had a disk in her data drive.

The moral of the tale is always have a disk in your default data drive when you work with the Ready chapter!

3 Extended Disk Systems

In this Section you learn how to use more than two 5.25-inch floppy disk drives with BEX. Part 1 provides the conceptual framework, explaining the *virtual drive number* system BEX uses to keep track of disk drives. Part 2 explains how you answer the configuration questions concerning extended disk drive systems. In Part 3 you learn the ins and outs of RAM drives, which can dramatically speed up your BEX use. Part 4 deals with 3.5-inch disks, sometimes called *microfloppies*, which are easy to slip in your pocket and can hold almost 800K of data. Part 5 deals with the Sider, the only hard disk system that BEX supports. Finally, Part 6 provides four sample combinations of extended disk systems.

Part 1: Assigning Virtual Drive Numbers

At the Learner and User Levels of BEX, your disk drive options were simple: one or two. Either way, they had to be 5.25-inch floppy disk drives, connected to a single disk controller card. Now at the Master Level, you have many more options: You can have more than two 5.25-inch disk drives; you can have one or two 3.5-inch disks drives; you can set aside portions of memory to be used as *RAM drives;* and you can use a Sider hard disk drive. We use the term *extended disk systems* to include all these options.

When DOS 3.3, the Apple operating system, addresses a disk drive, it uses the combination of the *slot* where the disk controller card is plugged in and

the *number* of the drive on that card. For the vast majority of Apple IIe systems, the first drive is slot 6, drive 1; the second drive is slot 6, drive 2. Even though the Apple IIc doesn't have slots, the same addresses are used: the internal disk drive is slot 6, drive 1; the external disk drive is slot 6, drive 2. Things are a little more complicated on the Apple IIgs because it often comes with 3.5-inch disk drives instead of or in addition to 5.25-inch disk drives--more on this in Part 4.

When you use a program, it would be awkward to always supply the full address of the disk drive, for example, "Give me chapter XYZ from the disk in slot 6, drive 2." And as you've learned using BEX, the program shortens the disk drive address to just one digit. At the Learner and User Levels of BEX, 1 means slot 6, drive 1; 2 means slot 6, drive 2. This is an example of a *virtual drive number.* The virtual drive number substitutes a shorter expression for the disk drive's full address. You enter a single digit, and the BEX program expands that to the full address that DOS 3.3 requires.

When you configure an extended disk system, BEX asks you to assign a slot and drive number to each virtual drive number. From then on, you access the disk in a particular slot and drive by the single virtual drive number you have assigned to that address.

Finally, before you configure an extended disk system, you need to know which drive the Apple boots from. Unless you have recently rearranged your disk drives or added new equipment, your Apple generally boots from the disk drive in slot 6, drive 1, which we refer to as the *booting drive.* You know this empirically, as you've been booting BEX successfully up to now.

For those with disk controllers in other slots, here's the nitty-gritty. The Apple IIe and IIc always attempt to boot from the first drive connected to the disk controller card in the highest numbered slot. When your disk drive controller is in slot 6, and slot 7 does *not* contain a disk controller card, the Apple boots from slot 6, drive 1. When you add another disk controller card with two drives in slot 5, the Apple still boots from slot 6, drive 1, because *6* is higher than *5.* In short, the Apple IIe and IIc boot from the highest drive number.

On the Apple IIgs, booting is determined by one Control Panel setting. When the Control Panel *Startup Disk* function is set to Scan, then the IIgs searches for a bootable disk in any slot. You can also specify any numbered slot or a

RAM or ROM drive for the startup disk. With BEX, you must set this *Startup Disk* function to your 5.25-inch floppy disk drive.

Part 2: Configuring an Extended Disk System

We'll demonstrate the general procedure with a straightforward example. Other configuration dialogues are shown in later Parts; Part 6 provides four sample combinations of various devices. Assume you have two 5.25-inch disk controller cards, one in slot 6 and one in slot 5. Each card has two disk drives plugged into it, for a total of four drives. After you have answered all the printer questions in a configuration, here's how the rest of the dialogue goes:

Do you have a extended disk system? Y <CR>
Virtual drive 1 is the Main program disk
For virtual drive 1
Enter slot: 6 <CR>
Enter drive: 1 <CR>
For virtual drive 2
Enter slot: 6 <CR>
Enter drive: 2 <CR>
For virtual drive 3
Enter slot: 5 <CR>
Enter drive: 1 <CR>
For virtual drive 4
Enter slot: 5 <CR>
Enter drive: 2 <CR>
For virtual drive 5
Enter slot: 0 <CR>
Enter a name for this configuration: FOURDISK

You may notice that this dialogue is similar to configuring printers. You assign a virtual drive number to the combination of slot and drive number. From then on, the virtual drive number is how you reference that disk drive. You end the naming process by entering zero for the slot number. BEX prompts you with the next virtual drive number in sequence until you supply eight addresses or enter zero.

Part 3: RAM drives

A RAM drive is a portion of the Apple's memory that acts like a floppy disk. *RAM* is an acronym for Random Access Memory. You can read and write information between a RAM drive and a floppy disk. A RAM drive is very similar in function to BEX's Ready chapter. All "disk" access happens in a flash, because you don't have to wait for the mechanical head on the disk drive to move around over a physical disk.

Caution! The most important thing to remember about RAM drives is this: Any information on a RAM drive disappears when you turn off the power! *When you want to keep a lasting version of your work, you must conscientiously copy chapters from the RAM drive to a physical disk drive.*

At the Master Level, you can configure RAM drives to hold data, and you can also configure one RAM drive to hold the programs on BEX's Main side. Things speed up dramatically when BEX runs from a RAM drive--moving to the Page Menu, for example, takes just as long as loading the Editor or Print options.

In order to configure a RAM drive, your Apple must contain more than 128K of memory. You get this extra memory by plugging a *memory card* into your Apple IIe or IIgs. Although the Apple IIc does not have slots, it's possible for a computer dealer to install extra memory in a IIc, as well.

- ***Two Kinds Of Memory Cards***

There are two distinct types of memory cards: the *auxiliary slot cards* and the *regular slot cards.* The difference between them is where you plug them in and how they are partitioned. BEX works with both kinds of memory cards.

The *auxiliary slot cards* are plugged in to the auxiliary slot of the Apple IIe. They provide 80-column capability as well as extra memory. As with any 80-column card, the Apple addresses an auxiliary slot card as if it were in slot 3,

even though it's not physically plugged in to slot 3. Some brand names are the RamWorks card from Applied Engineering and the MultiRAM card from Checkmate Technologies. Comparable cards are available for the Apple IIc, though it doesn't have slots. The Z-RAM card from Applied Engineering or the MultiRAM CX card from Checkmate Technologies can be installed in an Apple IIc. They act like an auxiliary slot card, and BEX treats them that way.

The Apple IIgs does not have an auxiliary slot; instead, it has a *memory expansion slot.* As far as BEX is concerned, a memory card plugged into the Apple IIgs's memory expansion slot is treated as if it were an auxiliary slot card in an Apple IIe. Apple IIgs users should follow the instructions for auxiliary slot cards.

The *regular slot cards* can be installed in slot 1 through slot 7 in the Apple IIe or IIgs. Examples of regular slot cards are the Apple Memory Card made by Apple, and the RamFactor card made by Applied Engineering. You can't use a regular slot card for DOS 3.3 RAM drives in a system that also contains a Sider hard disk.

Plain old DOS 3.3 does not have the capacity to create and work with RAM drives. However, it's possible to make minor disk operating system modifications, commonly called *patches,* that enable DOS 3.3 to work with RAM drives. BEX takes advantage of three different RAM drive patches. For auxiliary slot cards in the Apple IIe and Apple IIc, BEX uses RAMDRIVE, a program written and distributed by Applied Engineering. For expansion slot memory in the Apple IIgs, BEX uses RAM 3.3, a special patch written by David Holladay. For the Apple Memory Card in the Apple IIe or Apple IIgs, BEX uses patches located in the card's firmware. At the end of this Part, we discuss some of the idiosyncrasies you can encounter when working with regular slot memory cards.

- ***Installing and Checking Out Your Card***

For instructions on installing RAM cards, check Section 5, The Cookbook, in the Interface Guide. Turn off the power to your computer whenever you plug in cards! Don't try to install a card in an Apple IIc yourself. Have a trained technician install the card for you.

Once the card is installed, use option W - What is in this computer at the Starting Menu. When BEX says `# RAM drives available through slot 3` then BEX has recognized an auxiliary slot card. If BEX does not display this message, then it has not recognized your memory card. Check to make sure the card is properly installed.

When you install a regular slot card, What is in this computer should list it as an *Apple Memory Card* in its appropriate slot. When you install a regular slot card in slot 5, yet BEX says it's an "unknown card," do not worry. Option R - Recognition of cards on the Starting Menu lets you teach BEX about cards it does not recognize. You use Recognition of cards to tell BEX that the "unknown card" in slot 5 is actually a "Regular Slot Memory Card." More details in Interface Guide Section 15.

● *Dividing the Memory Into RAM Disks*

The memory on a regular slot card is treated as a single, large RAM drive. When you have a one megabyte Apple Memory Card in slot 5, its address for configuring purposes is slot 5, drive 1. Once configured, this RAM drive has *a lot* of room. You get `1480 sectors free` when you press the number sign at any menu.

Both Apple Engineering's RAMDRIVE software and RDC'S RAM 3.3 program divide the memory into 192K RAM drives, holding more than 700 sectors. How many RAM drives you get depends on how much memory is in the card. To find out exactly how many sectors are in each of your RAM drives, press # at any BEX menu. The maximum amount of memory that RAMDRIVE or RAM 3.3 can manage is one megabyte--five full RAM drives. Any additional memory above one megabyte is ignored.

As you can see, most RAM drives hold significantly more data than a 5.25-inch floppy disk, which starts out with 140K or 528 sectors free. Because of this difference in storage capacity, a RAM drive cannot be either source or target in option C - Copy disks on the Starting Menu. To copy information from RAM drives to more permanent storage media, use any creative BEX option like Copy chapters, Replace characters or Grade 2 translator, or use the FID utility available from BEX's Starting Menu. (More details on FID appear in User Level Section 13.)

- ***Configuring RAM Drives for Data***

As with any extended disk system, you must establish a BEX configuration that references the RAM drives. In the following sample, all the RAM drives on an auxiliary slot card are used as *data* drives.

Do you have a extended disk system? Y <CR>
Virtual drive 1 is for the program disk.
For virtual drive 1
Enter slot: 6 <CR>
Enter drive: 1 <CR>
For virtual drive 2
Enter slot: 6 <CR>
Enter drive: 2 <CR>
For virtual drive 3
Enter slot: 3 <CR>
Enter drive: 1 <CR>
For virtual drive 4
Enter slot: 3 <CR>
Enter drive: 1 <CR>
<beep> Error! Illegal duplication with virtual drive
For virtual drive 4
Enter slot: 3 <CR>
Enter drive: 2 <CR>
For virtual drive 5
Enter slot: 3 <CR>
Enter drive: 8 <CR>
<beep> Error! Enter a number between 1 and 5; or zero to cancel
For virtual drive 5
Enter slot: 3 <CR>
Enter drive: 3 <CR>
For virtual drive 6
Enter slot: 0 <CR>
Enter a name for this configuration: RAM <CR>

In this configuration, the booting drive and the program drive are both in slot 6, drive 1. The default data drive is drive number 5. Unless all the chapters you create are completely disposable, it is absolutely crucial that

you copy your chapters from a RAM data drive to a floppy disk before you turn off the power!

Also shown in this example are two of the error messages BEX provides when you enter nonsensical values. Remember, you can press <CR> for a summary of RAM drives available when BEX prompts for the slot number.

- ***Configuring a RAM drive as the Program Drive***

You can establish a configuration where the Main side of BEX is loaded to a RAM drive. If you have enough memory, you can have RAM data drives as well. When BEX is loaded on a RAM drive, moving between menus takes less than one second. Since you already have a floppy disk copy of your BEX program, loading the BEX Main side to a RAM drive requires less concentration on your part. It's like using an extra backup of your BEX disk. If lightning strikes and your Apple's power is affected, you won't be losing crucial data.

As we said in Part 2, BEX always looks for the program disk in virtual drive 1. When you load the Main side of BEX on the RAM drive, virtual drive 1 is not the booting drive. You can still boot BEX, because the Apple takes care of booting. Once BEX is booted and you enter the name of the configuration, BEX knows to look in virtual drive 1 for the Main side of BEX. Here's one possible arrangement for a 512K auxiliary slot card:

- Virtual drive 1: slot 3, drive 1
- Virtual drive 2: slot 6, drive 1
- Virtual drive 3: slot 6, drive 2
- Virtual drive 4: slot 3, drive 3
- Virtual drive 5: slot 3, drive 2

In this particular configuration, the default data drive is one of the two larger RAM drives. Notice that the virtual drive numbers do not have to follow the order of the full addresses of the disk drives. The program drive is always virtual drive 1, in this case, slot 3, drive 1. The booting drive is virtual drive 2, which is physically slot 6, drive 1. You could rearrange the virtual drives to suit your tastes; but to load BEX onto the RAM drive, you must assign virtual drive 1 to a RAM drive.

Once you set up a configuration with virtual drive 1 as a RAM drive, you're ready to go. With the Boot side *still in the disk drive,* press <space>. BEX checks to see if the Main side software is loaded on virtual drive 1. When it is loaded, you quickly move to the Main Menu. But when it's not loaded, BEX automatically uses FID to load the Main side software onto the RAM disk. At the appropriate time, BEX prompts you to insert the Main BEX disk and to press any key. When you need to access the Starting Menu, first go to the Main Menu and insert the Boot side of BEX into the booting disk drive, then press <space>. To get back to the Main Menu, press <space> at the Starting Menu. You can't load the Starting Menu on a RAM drive; you must access the Starting Menu from disk.

Preventing havoc when RAM drives share programs and data

Once you have loaded BEX onto the RAM drive, virtual drive 1 contains all the BEX program chapters *as well as* any data chapters you put there. When you have a regular slot memory card, which is treated as one vast RAM drive, then you definitely will be writing data chapters on your program drive. If you told BEX to perform an operation on *all* the chapters in virtual drive 1, you would change the BEX chapters `ZQFOR` and `ZQREV` which contain the translator tables. BEX recognizes a chapter by looking for a binary directory file that satisfies three criteria: Is it three sectors big? Is it unlocked? Is its second-to-last character *not* a period? You don't want to change the name of ZQFOR or ZQREV, and you can't make the directory file any bigger or smaller, but you can easily lock the directory file.

To prevent BEX from messing with any chapters on your Main side, lock their directory files *before* you install them on your RAM drive. You can use FID'S option 5 - Lock Files to lock every file on your Main side.

Caution!

Make sure to unlock any chapter from the Main side before editing: Quit BEX and type `UNLOCK FILENAME,S#,D# <CR>` using the appropriate slot and drive numbers. Then type `RUN <CR>` and edit your chapter successfully. When you edit a chapter and make any changes at all, BEX must change the information in its directory file. When you quit or move between pages, and the directory file is locked, BEX prompts: `Cannot write to disk, insert data disk in program drive and`

> `press any key` and saves the current page buffer in the `SAVE` chapter.

- ***Using RAM Drives***

BEX treats a RAM drive like any other data drive. When virtual drive 3 is a RAM drive, then editing chapter `3LETTER` creates a chapter named `LETTER` on drive 3. One big difference is that RAM drive access is totally silent. This can be a little disconcerting. When you have an auxiliary slot or memory expansion card, there is a *visual* indicator of disk drive access. You only see this when you use a non-HI-RES screen; in the lower right-hand corner, an inverse *W* shows writing and an inverse *R* shows reading.

You can add an audible indicator as well. When you boot BEX, hold down the open-Apple key until you hear a sound. For an Apple IIe, it's a clicking sound; for the Apple IIgs it's a low moan. When you do this, you add an audible indicator for RAM drive access. You hear a quick clicking whenever you read from or write to the RAM drives. No visual or audible indicator is available for the RAM drive from a regular slot card.

The crucial thing to remember is you lose your data if you lose power. *You must copy your chapters to disk before turning off your computer.* In fact, we recommend that you routinely copy your chapters to disk at least every half-hour. In Section 4, Part 3, we explain how you can catalog more than one disk drive at the Page Menu. When you configure your RAM drives all in a row, use this feature to check to make sure you've saved all your RAM drive data.

You can use almost every BEX option with a RAM drive, with three exceptions:

- 1. You cannot read a ProDOS textfile from a ProDOS RAM drive. The Second Menu's option R - Read textfiles can only read ProDOS textfiles on 5.25 inch or 3.5 inch disks. You *can* read and write DOS 3.3 textfiles to and from the RAM drive.
- 2. You cannot initialize a RAM drive. Option I - Initialize disks on the Starting Menu refuses to initialize a RAM drive. If you typed `INIT HELLO` at the BASIC prompt, you would "crash" the RAM drive, and you would have to reboot BEX. When you have an auxiliary slot card or

memory expansion slot card, you can erase all the contents of all RAM drives. As BEX boots, it loads the RAMDRIVE or RAM 3.3 software. When you depress the solid-Apple (or Option) key as BEX boots, the contents of all RAM drives are wiped clean.

- 3. You can't copy the contents of an entire RAM drive with option C - Copy disks on the Starting Menu. Use any creative BEX option, or FID, to copy information from a RAM drive to a more permanent storage medium.
- You can reboot BEX without affecting the information on the RAM drive. (If you hold down the solid-Apple key when booting, though, all the information on the RAM drive is wiped clean.)

Hint!

> If you have a *regular slot* memory card, you can write DOS 3.3 chapters or textfiles to it, then quit BEX. Without turning off the power, start up QTC. You can then tell QTC that your source disk is in slot 4, drive whatever, and QTC quickly converts the DOS 3.3 BEX chapters on the DOS 3.3 RAM drive to ProDOS textfiles on disk. This won't work with an auxiliary or memory expansion slot card.

Redirecting virtual drive 1 at the Starting Menu

There's one additional complication when you run the Main side from a RAM drive. As we've stressed, virtual drive 1 is always the program drive. On the Main side, the programs BEX needs are on the RAM drive you've configured as virtual drive 1. But when you switch to the Starting Menu, the programs BEX needs are on the floppy disk in the booting drive. When you use the Starting Menu, virtual drive 1 is temporarily redirected to the booting drive. When you <space> back to the Main Menu, virtual drive 1 becomes the Main RAM drive again. As mentioned in Part 2, there's an on-line reminder of which virtual drive number references which physical disk drive. Press D at any BEX menu; when BEX responds `Which drive? #` enter question mark followed by <CR>. When you do this at the Starting Menu, you may press M to catalog the Main side RAM drive.

The 105 filenames barrier and regular slot cards

Two factors limit how much data can fit on a DOS 3.3 disk of any sort: the free sector count, and the number of filenames. Clearly, the free sector count is not a problem here; a regular slot memory card gives you *lots* or room. However, plain DOS 3.3 can only manage 105 filenames per "disk." Because

the auxiliary and memory expansion card RAM drives are subdivided into drives, you usually don't bump into this 105 filename barrier. But on a one megabyte regular slot card, you're bound to encounter it sooner or later.

When you attempt to save the 106th file, DOS reports the `DISK FULL` error message, even if you have loads of sectors free. Each BEX chapter is composed of several files: one binary file for each BEX page, plus one more binary file for the chapter directory.

And this is where you can get into trouble. Suppose you have five 20-page BEX chapters. They are stored as 105 files on disk. Even if each of the pages of these five BEX chapters contained only one character, you would get a `DISK FULL` error message if you tried to save another BEX chapter to this disk. You must delete some files from the RAM drive to free up filename slots for a new chapter.

The limit of 105 filenames is particularly problematic when you have a regular slot memory card, and you load BEX's Main side onto a RAM drive. The BEX Main programs are at least 35 files; this leaves just 70 possible filename slots for your BEX chapters.

Can't write to disk errors in the Editor and RAM drives

Way back in Learner Level 5, we described how BEX attempts to salvage the page buffer when something goes wrong when you quit or move between pages. BEX uses the `RUN 999` technique to save the page buffer in a one-page BEX chapter named SAVE *on your program drive.* Suppose you were editing one of the five 20-page chapters mentioned above. You enter control-C control-P to cut pages. You have just asked BEX to create another page file, but DOS 3.3 refuses to accept this 106th filename. Because BEX has received a DOS error, it swings into action, attempting to create a SAVE chapter--but DOS 3.3 refuses again. In this situation, BEX tells you `Insert data disk in program drive and press any key` *endlessly.* But you can't insert a different data disk, because it's a RAM drive. Here's how you recover:

- 1. Press control-Reset
- 2. To restore speech output, enter `PR#0 <CR>`
- 3. Type `CATALOG <CR>`

- 4. Decide what files you can live without, and kill them off with the `DELETE` command. As a last resort, you can kill off the translator chapters, ZQFOR and ZQREV, and then later recopy them from your Main disk.
- 5. Now, you can type `RUN 999 <CR>` and BEX can create the SAVE chapter.

Part 4: 3.5 Inch Disk Drives

The designers of DOS 3.3 never anticipated a device like the 3.5 inch disk drive. This is a pity, because these little disks are delightful: they are compact, sturdy, and they can hold almost 800K of data. But DOS 3.3 can never cope with more than 400K on one disk. Fortunately, Gary Little wrote a modified disk operating system named AmDOS, which allow access to 3.5 inch disks. Even more fortunately, Mr. Little licensed us to include AmDOS on the BEX disk. The AmDOS on the BEX disk has been slightly modified to work smoothly within BEX.

Caution! The BEX program disk cannot be loaded onto a 3.5 inch disk. This means your Apple system must have at least one 5.25 inch disk drive in order for you to use BEX.

Refer to the Interface Guide, Section 1 on the Apple IIgs for information on connecting 3.5 inch drives to your system. If you are going to do a lot of work with 3.5 inch disks, you may want to purchase a full featured version of AmDOS at $20 US--check Appendix 5 for the address.

AmDOS gets around the 400K limitation in DOS 3.3 by dividing each 800K disk drive into two 400K disks. (It's yet another virtual drive scheme!) Even though the two 400K disks are physically on the same magnetic medium, they are treated as if they were two totally separate disks. The tricky part is how these 400K disks are addressed: both drive numbers on the first disk drive are always odd, and on the second disk drive they're always even.

Suppose your 3.5 inch disk controller card is plugged into slot 5. AmDOS treats the first 3.5 inch disk drive as if it were two distinct 400K drives: one in

slot 5, drive 1 and the other in slot 5, drive 3. When you have a second 3.5 inch disk drive, then AmDOS addresses that 800K disk as two 400K disks, one in slot 5, drive 2, the other as slot 5, drive 4. This system has two potentially confusing aspects: one physical disk drive is divided into two addresses; and, the drives are not numbered sequentially. Fortunately, you only have to supply the full address to BEX when you configure; from then on, you can use BEX's single virtual drive number.

Apple IIgs: When you have two 3.5 inch disk drives, you must modify a Control Panel setting to access the second disk drive. The bottom item on the Control Panel is RAM Drives: set both the minimum and maximum size to zero. You must turn off the power for this change to take effect. If you don't do this, then the IIgs will hide the second 3.5 inch disk drive.

- ***Configuring BEX with 3.5 Inch Disk Drives***

Let's say you have two disk drives in your system: one 5.25 inch floppy drive in slot 6, drive 1, and one 3.5 inch drive in slot 5, drive 1. You could configure this way:

Virtual drive 1: slot 6, drive 1

Virtual drive 2: slot 5, drive 1

Virtual drive 3: slot 5, drive 3

In this sample, your default data drive is virtual drive 3, one half of the disk in the 3.5 inch disk drive. And that information is *always* going to be on that half of the 3.5 inch disk. Suppose you established a different configuration that reversed the virtual drive numbers for the 3.5 inch disk, like this:

Virtual drive 1: slot 6, drive 1

Virtual drive 2: slot 5, drive 3

Virtual drive 3: slot 5, drive 1

You write a chapter on drive 3 in the first system. To print that same chapter with the second system, you look on drive 2.

- ***Initializing and Using 3.5 Inch Disks***

Once the AmDOS software is loaded into the Apple's memory, you can no longer initialize 5.25 inch floppy disks. So, it's important to understand *when* BEX loads the AmDOS software. There are two Starting Menu actions that make BEX load the AmDOS software:

- 1. Initializing a 3.5 inch disk - or
- 2. Moving to the Main Menu with <space>

You initialize a 3.5 inch disk using option I - Initialize disks. When you supply a virtual drive number corresponding to the 3.5 inch controller card, BEX realizes that you need AmDOS, and loads it. With the sample configuration above, after you start initializing, entering 2 or 3 at the `Which drive?` prompt loads AmDOS.

Caution!	AmDOS initializes both 400K drives at the same time. In our example here, initializing either drive 2 or drive 3 results in initializing both of them.

Once you have initialized a 3.5 inch disk, you can no longer initialize 5.25 inch disks until you reboot. You can't copy 5.25 inch disks either, because the first step in BEX's Copy disks is to initialize the target disk.

The other action that loads AmDOS is when you move from the Starting to the Main Menu. When you have done either of these actions, you simply can't initialize 5.25 inch disks. Therefore, before you begin using a configuration that includes 3.5 inch disks, sit down and initialize a stack of floppy disks! If you must initialize a floppy disk after AmDOS is loaded, then save your data and reboot BEX.

Can't copy entire disks

Besides the initializing disks issue, there's just one limitation on BEX use with 3.5 inch drives. You can not use option C - Copy disks to make copies of 3.5 inch AmDOS disks. (You can only use Copy disks for 5.25 inch floppies

before you load AmDOS.) To get material on or off a 3.5 inch disk you must use the normal machinery of BEX (copy chapters, translation, replacing, etc.) or use FID. To use FID, AmDOS must be already loaded. All you need to do is move to the Main Menu and then move back to the Starting Menu and press F.

Textfiles and 3.5 inch disks

Option R - Read textfiles to chapters on BEX's Second Menu can read ProDOS textfiles stored on ProDOS 3.5 inch disks. With our sample, you would insert the ProDOS disk in the 3.5 inch drive, then enter either virtual drive number 2 or 3 when BEX prompts for the textfile to read. You can Read the textfile to a RAM drive, to the Ready chapter, or to a DOS 3.3 floppy disk. As mentioned in User Level 10, however, BEX can only read ProDOS textfiles that are stored at the *root* level of a ProDOS volume. If the 3.5 inch ProDOS disk was named `/LETTERS` BEX can read the textfile named `/LETTERS/SANDY` on that disk. But BEX cannot read the textfile named `/LETTERS/JUNE/SANDY` because the file `SANDY` is stored in the subdirectory named `/LETTERS/JUNE/`

Part 5: Sider Hard Disk

The Sider hard disk, manufactured by First Class Peripherals, is the only hard disk system supported by BEX. You can only configure a single Sider with BEX; BEX won't be able to work with additional Siders *daisy chained* on to the first Sider. You can configure an extended disk system that includes both auxiliary card RAM drives and the Sider. However, you can not load the Main side of BEX onto the RAM drive when your configuration includes the Sider.

With DOS 3.3, you cannot install a regular slot memory card and a Sider in the same system. The Sider disk controller card contains special patching software to work well with DOS 3.3. The regular slot memory card also patches DOS 3.3. Unfortunately, the DOS 3.3 patches are *different*, yet they occur at the same point in DOS. Section 13 of the Interface Guide is devoted

to extensive installation instructions for the Sider, including tips on partitioning the Sider into different operating systems.

The DOS 3.3 partition of the Sider is further subdivided into numbered *volumes.* When you set up the Sider, you establish some number of *small* volumes containing approximately 140K or 528 sectors, as well as some number of *large* volumes containing approximately 400K or 1536 sectors. (Make sure you write down how many volumes you've created; you need the total number when you configure.) You install the Sider disk controller card in slot 7. The *full* address of Sider's volume 28, for example, is slot 7, drive 1, volume 28.

However, when you configure the Sider, you do not establish a distinct virtual drive number for each Sider volume. Instead, you enter one virtual drive number for the slot and drive of the Sider's disk controller card, and then provide the total number of DOS 3.3 volumes. You also provide the volume numbers for the Boot and Main sides of BEX.

- ***Configuring the Sider***

Once BEX has the Sider volume numbers, it automatically generates a list that pairs a virtual drive number with the Sider's volume number. Since the BEX Main program is contained on the Sider, you must make the Sider virtual drive 1. Since the Sider is usually installed in slot 7, virtual drive 1 is slot 7, drive 1. Here's the end of a sample configuration dialogue, after you've answered all the printer questions:
Do you have an extended disk system? Y <CR>
Virtual drive 1 is for the Main disk program disk.
For virtual drive 1
Enter slot: 7 <CR>
Sider Hard Disk
How many volumes: 48 <CR>
Boot side volume: 2 <CR>
Main side volume: 3 <CR>
For virtual drive 49
Enter slot: 6 <CR>
Enter drive: 1 <CR>
For virtual drive 50

Enter slot: 6 <CR>
Enter drive: 2 <CR>
For virtual drive 51
Enter slot: 0 <CR>
Enter a name for this configuration: SIDER <CR>

For the question about how many volumes, use the number from the Sider initialization (the sum of the small and large DOS 3.3 volumes). For the questions about the boot side volume and the main side volume, give the volume numbers where you copied the program disks. From that point on, you answer the virtual drive number questions as always.

- ***Using the Sider***

Whenever you boot from the Sider, you are presented with the Sider's Master Menu, which has seven choices. The default choice is number 3, Boot from DOS partition. Press <CR>, and you run the DOS 3.3 `HELLO` program. Following the instructions from the Interface Guide, you have modified this HELLO program to run BEX. So after you press <CR> at the Sider's Master Menu, the next thing you hear is your old friend BEX saying `Enter configuration:`

Moving between the Main and Starting Menus is the same as always: press <space>.

BEX treats each volume in the Sider as a separate disk drive. In the sample configuration, the default data drive is a floppy drive, virtual number 50. Virtual drives 2 through 48 are all Sider volumes, virtual drive 49 is also a floppy disk drive. As always, virtual drive 1 is the program disk. When you're at the Main, Second, or Page Menus, virtual drive 1 is redirected to the Sider volume containing the Main programs. When you're at the Starting Menu, virtual drive 1 is redirected to the Sider volume contains the Boot programs. See Part 3 of this Section, *Redirecting Virtual Drive 1 at the Starting Menu* for an explanation of what's going on.

Of course, it may take some time to get used to having many megabytes of storage on a hard disk at your disposal. Automatic procedure chapters, discussed in Section 8, are particularly handy tools when combined with the

Sider. Section 4, Working with Chapters, discusses some Master Level features that help you keep track of all those Sider volumes.

- ***Operating System Differences***

BEX uses a patched DOS 3.3 that speeds up disk access. The Sider uses a more conventional version of DOS 3.3. You will notice that some accesses to floppy disks are slower on the Sider. On the Sider, you do not get the free sector count at the top of a catalog. But you can enter number sign at any BEX menu to obtain the free sector count on any of your virtual drives.

Part 6: Four Sample Configurations

The following four samples are based on Apple systems we use here at Raised Dot Computing. You will notice that, where possible, we load the Main side software on a RAM drive. This maneuver so speeds up the system that it's irresistible.

MB Apple IIe with four virtual drives

Combining a regular slot memory card with two floppy disks, this set-up is excellent for massive data manipulation with Contextual Replace. With a one megabyte card, you can copy BEX to the RAM drive and have plenty of room left for your data.

Virtual Drive 1: slot 5, drive 1 (Apple Memory Card)

Virtual Drive 2: slot 6, drive 1 (floppy drive 1 and booting drive)

Virtual Drive 3: slot 6, drive 2 (floppy drive 2)

MB Apple IIgs with eight virtual drives

The one megabyte of memory in the IIgs' memory expansion slot is divided into five RAM drives, attached to slot 3. A single 3.5 inch disk drive becomes two virtual drives; the 5.25 inch drive is basically only used to boot the

system. In this configuration, the booting drive and the default data drive are the same.

- Virtual Drive 1: slot 3, drive 1 (RAM drive 1)
- Virtual Drive 2: slot 3, drive 2 (RAM drive 2)
- Virtual Drive 3: slot 3, drive 3 (RAM drive 3)
- Virtual Drive 4: slot 3, drive 4 (RAM drive 4)
- Virtual Drive 5: slot 3, drive 5 (RAM drive 5)
- Virtual Drive 6: slot 5, drive 1 (3.5 inch drive, first half)
- Virtual Drive 7: slot 5, drive 3 (3.5 inch drive, second half)
- Virtual Drive 8: slot 6, drive 1 (floppy drive and booting drive)

521K Apple IIe with hard disk

In this configuration the Sider comes first, then the RAM drives, and then the floppy drives. You cannot load the Main side software onto a RAM drive when you boot from a Sider.

Virtual Drive 1: slot 7, drive 1 (Sider Hard Disk, booting and program drive)

Number of volumes: 43

Boot side volume: 2

Main side volume: 3

Virtual Drive 44: slot 3, drive 1 (RAM drive 1)

Virtual Drive 45: slot 3, drive 2 (RAM drive 2)

Virtual Drive 46: slot 3, drive 3 (RAM drive 3)

Virtual Drive 47: slot 6, drive 1 (floppy drive 1)

Virtual Drive 48: slot 6, drive 2 (floppy drive 2)

512K Apple IIc with four virtual drives

The Z-RAM from Applied Engineering acts just like an auxiliary slot memory card. With the Main side loaded in RAM, you only need to use the internal disk drive. This makes the Apple IIc reasonably portable.

Virtual Drive 1: slot 3, drive 1 (RAM drive 1)

Virtual Drive 2: slot 3, drive 2 (RAM drive 2)

Virtual Drive 3: slot 3, drive 3 (RAM drive 3)

Virtual Drive 4: slot 6, drive 1 (internal floppy drive and booting drive)

4 Specifying Chapters

With the many drives that are available at the Master Level, knowledge of how to specify chapters and alter chapter names becomes all the more important. In this Section we discuss the many ways of specifying chapters. In Part 1 we review methods of chapter selection introduced in User Level Section 4, Part 1. In Part 2 we introduce two new target chapter naming methods. In Part 3 we discuss two methods of managing your data.

Part 1: The Chapter Prompt

Like most prompts at the Master Level, the prompts for chapters are quite short. BEX simply asks for `Chapter:` or `Target chapter:` or `Naming method:` For selecting chapters, all the User Level possibilities are available, with several features unique to the Master Level. We have already discussed how you can answer the chapter prompt with the right bracket to indicate that you are using the Ready chapter. You can have more than two disk drives at the Master Level. This means that you can answer the chapter prompt with `7CHAPTER` or `8CHAPTER` assuming that you have that many drives.

Whenever you type a chapter name or naming method, BEX assumes you're writing and reading on the *default data drive.* When you are using an extended disk system, the disk drive numbers you use at chapter prompts are the virtual drive numbers you established in your configuration. The highest virtual drive number is always your default data drive number.

When you have configured with four drives, your default data drive number is 4.

- ***The Ready Chapter***

You can indicate `]` for the Ready chapter at all but one chapter prompt: option N - Name change for chapters on the Second Menu. When you specify the Ready chapter with Fix chapter directory, BEX creates 4095 character pages for each Ready page in memory. See Section 2 for further explanations.

When you enter `]` at many `Chapter:` prompts, BEX recognizes the Ready chapter. But BEX does not recognize `]` as a chapter name after it has given a numbered list for a particular drive. When you're choosing numbers from a list, then `]` at the `Chapter:` prompt is just like pressing <CR> alone.

- ***Summary of Chapter Selection***

This chart summarizes the chapter selection features available at the Master Level:

- `NAME` - chapter NAME on default data drive (when you have an eight drive system, this is drive 8)
- `5NAME` - chapter NAME on drive 5
- `2` - scan drive 2 and present numbered list of chapters
- `/` - scan highest (default) data drive; when 8 is your highest numbered drive, then BEX scans drive 8
- `5/Q` - scan drive 5 for chapters ending in *Q* and present a numbered list
- `+3` - scan drive 3, present numbered list, then reprompt for more chapters
- `+7/2` - scan drive 7 for chapters ending in 2 and reprompt for another chapter selection
- `]` - The Ready chapter

Part 2: Target Chapter Naming Methods

All the target chapter naming methods described in the User Level are available at the Master Level. In addition, you can always use the Ready chapter as a target chapter (except at option N - Name change for chapters). In fact, you can specify the Ready chapter when you use I to individually name the target chapters. At the Master Level, there are two additional naming method options: drive number 0 and the period prefix.

- ***Disk Number Zero***

Using drive number 0 in a target chapter naming method directs each target chapter to its *home drive,* the same drive as the source chapter generating it. For example, suppose you specify source chapters on both drive 2 and drive 3 and use naming method 0S. Each source chapter generates a target chapter by the same name on the same drive and thus overwrites itself. Drive number 0 is always available when you are naming one or more target chapters. However, it is most useful when you have specified source chapters from more than one drive.

- ***The Period Prefix***

The period prefix enables you to use a target chapter naming method even if you have given only one source chapter. When you have more than one source chapter, the period has no effect, and BEX operates as if you had specified a naming method without the period. The period character goes after the drive number, if present, and before the letter that specifies the naming method. It tells BEX, "This is a naming method, not a chapter name." For example:
Chapter: MASTERPIECE
Chapter: <CR>
Target chapter: 6.S

The result is a chapter named MASTERPIECE on drive 6.

This means that you have to type a chapter's name *only once* in its life: when you create it. After that you can always specify it by number, and modify its name using a target chapter naming method.

Limitations of the period prefix

You *cannot* use the period prefix to name a target chapter in a list of individually named target chapters. When you use target code I, you must type each target name. For example, suppose you are copying chapters from drive 3, a RAM drive, to drive 8, on disk. You have a list of three chapters, and use the I naming method:
Main: C
Copy chapters
Chapter: 3 <CR>
There are 3 chapters:
1 BIOPAPER
2 ESSAY2
3 CHEM NOTES
Use entire list? N Y <CR>
Naming method: I <CR>
For chapter BIOPAPER
Target chapter: 8.S <CR>
For chapter ESSAY2
Target chapter: 8.A-2 <CR>
For chapter CHEM NOTES
Target chapter: 8LAB NOTES <CR>
BEX begins to copy the chapters, then crashes with a `SYNTAX ERROR` message. BEX tried to name the BIOPAPER chapter `.S` and DOS 3.3 won't let a filename start with a period. At this point you type `RUN <CR>` and copy the chapters again, typing out the names.

Section 7 discusses using the period prefix in automatic procedure chapters. The period prefix enables the same procedure to work with a list containing one chapter or with a list containing many.

- ***Target Codes***

This chart summarizes the possibilities:

- `3AXYZ` - add characters *XYZ*, write chapters on drive 3
- `LXYZ` - remove the last character, then add the characters *XYZ*, write on default data drive
- `7.S` - same name, write on drive 7
- `0D` - delete last character, write target chapters on "home" drives
- `.I` - individually name target chapters (and specify drive numbers, if desired)

- ***Combining Scanning Methods and Target Codes***

Here's an example to show what you can do when you combine scanning methods and target codes: Suppose you have chapters on three separate drives, all of which you need to use Replace characters on, and you want to keep them on the same drives. Ordinarily, you would use Replace characters three separate times, once for each drive. But by using a combination of the plus sign when scanning for chapters, and drive number 0 when specifying the target chapter naming methods, you can do it in one step.

For this example, we use three drives: drives 1 and 2 are RAM drives, and drive 3 is a 5.25 inch disk drive. There are two chapters on each drive.

First, you scan each of the three drives for the chapters that need replacing, by using the plus sign before the drive number to tell BEX to select more chapters. You choose the chapters from the numbered list, then move onto the next drive, until you have specified all the chapters. Next, BEX asks for a target chapter naming method. You could use any naming method, but you use the S naming method, to make things simple.

Here's how the dialogue looks:
Main: R
Replace
Chapter: +1 <CR>
There are 2 chapters:
1 PURPLE
2 RED
Use entire list? N Y <CR>
Select more chapters
Chapter: +2 <CR>

There are 3 chapters:
1 GREEN
2 BLUE
3 BROWN
Use entire list? N <CR>
Select chapters by number
Chapter: 1 <CR>
GREEN
Chapter: 2 <CR>
BLUE
Chapter: <CR>
Select more chapters
Chapter: 3 <CR>
There are 2 chapters:
1 YELLOW
2 BLACK
Use entire list? N Y <CR>
Naming method: 0S <CR>
Use transformation chapter: 1REPLACE <CR>
Continue? Y <CR>
Chapter PURPLE done
Chapter RED done
Chapter GREEN done
Chapter BLUE done
Chapter YELLOW done
Chapter BLACK done
Replaced 50 times

Part 3: Two Features With Whole Disk Catalog

Since you can have many disk drives at the Master Level, you need more tools to help you manage data. Two features help you with option W - Whole disk catalog on the Page Menu.

- ***Catalog More Than One Drive***

When you press W for the Whole Disk Catalog on the Page Menu, you can obtain a catalog of a range of disk drives. This feature makes it easier to keep track of the many volumes in a Sider hard disk, or of chapters in RAM drives. This feature is available at all Levels, but is most useful when you have more than two disk drives.

As when you're specifying chapters on more than one drive, you use the plus sign + to catalog more than one drive. When you press W at the Page menu, you're prompted: `Which drive?` followed by your default data drive number. Answer with a plus sign followed by a drive number followed by <CR>:

```
Page Menu: W
Whole disk catalog
Which drive? 8 +6 <CR>
through:
```

At this prompt, you enter the number of the highest drive you wish to catalog:

```
through: 8 <CR>
Disk drive 6
501 free sectors
There are 2 chapters on this disk:
CHECKS 1 pages 322 size
ADDRESS 1 pages 86 size
Grand total 408 characters
Disk drive 7
398 free sectors
There are 2 chapters on this disk:
DIARY 2 pages 8122 size
PAPER 2 pages 4586 size
Grand total 12708 characters
Disk drive 8
456 free sectors
There are 1 chapters on this disk:
DATA 4 pages 11529 size
Grand total 11529 characters
```

You must enter the drive numbers in ascending order; if you entered `+3 <CR>` through `1 <CR>` you would receive a `Bad range` error message.

- ***Save Whole Disk Catalog***

Using ampersand `&` before the drive number when specifying drives with option W - Whole disk catalog on the Page Menu allows you to save the screen output of the Whole disk catalogs in a BEX chapter. When you are prompted `Which drive?` enter an ampersand `&` followed by a drive number, then press <CR>. The ampersand, like the plus sign, allows you to scan multiple drives. It invokes the `through:` prompt and you enter the drive number of the highest drive you wish cataloged. When the catalogs are complete, you are prompted for a chapter name. Here is an example:

Page Menu: W
Whole disk catalog
Which drive? &2 <CR>
through: 3 <CR>
Disk drive 2
87 free sectors
There are 3 chapters on this disk:
Q-SAVE 4 pages 7566 size
WW 2 pages 4956 size
LETTER 6 pages10745 size
Disk drive 3
502 free sectors
There are 1 chapters on this disk:
REPLACE 1 pages 54 size
Grand total 26227 characters
Save list as
Chapter: CAT <CR>

Chapter CAT now contains all the characters that appeared on the screen between entering `2 <CR>` and the `Chapter:` prompt. This chapter cannot contain more than 2048 characters.

5 Printing

Master Level printing features focus on some of the finer distinctions in your output. In this Section, we describe embedding $$ commands in words, controlling how lines break, and touch on embedding printer control codes in your chapter.

Part 1: The Special Spacing Commands

BEX's formatter divides your text into output lines. Before this, the lines always broke at the <space> or <CR> that defines a BEX word. <Control-S>, the sticky space token, lets you define a space that is not a word boundary. <Control-T>, the touching token, can replace the initial and final space in BEX $$ format commands.

- ***Activating the Special Spacing Commands***

Some printers use <control-S> and <control-T> as control codes; you must tell BEX when to pay attention to these two characters in your text. To have BEX recognize them, you must first activate the commands with a special format command: $$ss activates the sticky space token and the touching token (see below) for the rest of the print stream. To disable special spacing, use $$d or reload the print program. If you have entered the sticky space token and the touching token into your text but you did not enter $$ss also, no space appears between the characters surrounding the tokens.

You may want to use <control-S> or <control-T> as printer control codes in your chapter. When you do, place the $$ss after where the codes are entered in your text. Before the $$ss, the formatter sends the <control-S> and <control-T> codes through to your printer.

- ***The Touching Token: <Control-T>***

The <control-T> *touching token* allows you to omit the space before a $$ command, and still have that command executed. When enabled by $$ss, a <control-T> in your text allows you to embed format commands within words. Like a space or <CR>, it can introduce a format command to be recognized and executed by the formatter. The touching token is particularly useful for tricky underlining, and for superscripts and subscripts. However, the formatter does break a line at a touching token.

For example, to underline *War and Peace* inside of parentheses, type:
(<control-T>*War and Peace*<*control-T*>)

Recall from User Level Section 7, Part 8, that suppressing underlining before a final period, comma, semicolon, or colon happens automatically when enabled by $$sp, and does not require any fussing with <control-T>.

The Grade 2 translator treats any control character as a space. This is important to keep in mind when you are embedding BEX $$ commands within words and intend to translate this text. Suppose you want to emphasize the third syllable in the word *computerized:*
comput<control-T>$$eb<control-T>er<control-T>$$ec<con trol-T>ized

Since the <control-T> takes the place of the spaces required before and after any $$ command, this word prints with the third syllable in boldface. When you translate this word, the final <control-T> separates the syllable *er* from the format command $$eb. This separation turns the translator back on, so the translated result is
-put]iz$
But if you don't end the $$eb with <control-T> the translator is off for all the characters after the $$ until the next space. When your print is:
comput<control-T>$$eber<control-T>$$ecized
then the translator creates:
-mputerized

- ***The Sticky Space Token: <Control-S>***

The normal use of the <control-S> *sticky space token* is as a *nonbreaking space.* Enabled by $$ss, a <control-S> in your chapter is printed as a space in your output. However, the <control-S> is not treated as a word delimiter for printing. Text connected by sticky spaces is treated as one BEX word and is all printed on the same line. In other words, a <control-S> creates a space in your output that is not a space in your text.

For example, suppose a character in the story you're writing has three middle initials. It would be confusing if his name was broken between lines. when you type
General<space>Montgomery<control-S>
T.<control-S>M.<control-S>J.<control-S>

Hambone
then his full name is never broken between lines.

- ***Writing about BEX $$ Commands Using <Control-S>***

Using a program to write about itself is always a little tricky. When we write about BEX format commands, we use the sticky space token. A <control-S> sticky space before the dollar sign in a format command is not a *real* space. So the format command or indicator is not executed; it is appropriately spaced as an individual word in the text.

For example, when we wrote about the underline command in User Level, Section 7, we printed the command by typing: <control-S>

The sticky space token also makes it possible for us to write about the paragraph token with its parentheses: ($p) is typed in the text as eight keystrokes: (<control-B> S $ p <control-B> S)

Remember that each <control-S> takes two keystrokes: control-C then S.

Part 2: Discretionary Hyphens and Linebreaks

These control characters provide the formatter with ways to break a line in the middle of a word.

- ***The Discretionary Hyphen: <ASCII 31>***

Use the discretionary hyphen to tell the formatter where to place a hyphen if it breaks that word between lines. If BEX doesn't break the word between lines, it won't print the hyphen.

To enter a discretionary hyphen in your text, hold down the control key and press the hyphen. In a HI-RES screen mode, a discretionary hyphen is represented in the Editor by a small, uppercase *S* with a vertical line down

the right side and a horizontal line underneath: much like the <ESC> character in your text. When you arrow over a discretionary hyphen, the Echo says "ASCII 31," which sounds like "As-key three one."

You can enter a discretionary hyphen manually in extra-long words in the Editor, or you can write a transformation chapter to insert a discretionary hyphen at suitable syllable boundaries. If you write a transformation chapter, be cautious. If you asked to insert <ASCII 31> before *ation* for example, you'd get discretionary hyphens in *n-ation* and *st-ation,* as well as longer words. Also, this works best for print chapters.

Caution! We don't recommend translating chapters containing this control character, as the translator treats <ASCII 31> as a space. If you typed `know<ASCII 31>ledge` then you get `"k<ASCII 31>l$ge` instead of `k` as you should get. Take out the discretionary hyphens before translating.

You can created an auto chapter that replaces <ASCII 31> with nothing and then runs the translator. See Section 7 for instructions about creating Auto chapters.

- ***The Discretionary Linebreak: <ASCII 30>***

The discretionary line break tells the printer that it's okay to break a line at that particular spot. For print output, this is a useful command to place after hyphens, dashes, and slashes that already appear in the text. For example, the paragraph above contains the word *extra-long.* Placing a discretionary line break after the hyphen allows the line to break there without printing an *extra* hyphen.

Hold down the control key and press the number `6` to enter a discretionary line break in your text. In the Editor, a discretionary linebreak is represented by a small uppercase *R* with a vertical line down the right side, and a horizontal line underneath: like the discretionary hyphen and the <ESC> character. The Echo speaks a discretionary linebreak as "ASCII 30", which sounds like "as-key three zero."

You only have to do this in chapters for print output. BEX's Grade 2 translator automatically places discretionary line breaks after hyphens and before and after dashes when it translates.

- ***Activating Discretionary Hyphens and Linebreaks***

For either of these control characters to work, you have to activate them at the beginning of your chapter with a $$sd command. This is just like using $$ss to activate sticky spaces. However, you may want to use <ASCII 30> or <ASCII 31> as printer control codes in your chapter. When you do, place the $$sd after where the codes are entered in your text. Before the $$sd, the formatter sends the <ASCII 30> and <ASCII 31> codes through unmodified.

In braille chapters, you do not need to enter $$sd. All output devices configured as *braillers* automatically include $$sd when printing.

Part 3: Repeat a Character

The format command $$vrX repeats the character *X* to the end of the current line (as defined by whatever margins are in effect). *X* may be an underline, dash, the space character, or any other printable character. Only one $$vrX command works on any one line. Use a hard <CR>, or a paragraph ($p) or a new-line ($l) idicator to termiate the line properly. You may place other characters on the line using the $$t# tab or $$p# positioning commands.

For example, a form requires the word *City* at the left margin, the word *State* two-thirds of the way across the line, and the word *ZIP* almost at the right margin. Wherever words don't appear, you want underlines. With a carriage width of 65, these commands would do the trick:
City $$vr_ $$p40 State $$p53 ZIP <CR>

- ***Kram a Word***

Kram one BEX *word* on the right margin: $$vk is used in combination with $$vrX in tables of contets or menus. For example:
Ham and Eggs: $$vr. $$vk $1.00 <CR>

You can use the sticky space token if you want more than one word at the end of the line. For example:
Ham and Eggs: $$vr. $$vk $1.00<control-S>(Fridays<control-S>Only)<CR>

The sticky space makes the three words into one BEX *word.* They appear as three separate words in the printed text. There is a limit, however. The word you have crammed to the right margin with $$vk cannot be more than 32 characters

---

Part 4: Embedding Printer Control Commands in Your Text

Many printers allow you to print fun things like boldface fonts, superscripts, subscripts and foreign language letters; you can change line spacing, use fancy vertical tabs, and turn on headline mode. Diligent study of your printer manual will yield what characters you need to enter to print these wonders. Since you can type any ASCII character in a chapter, it's easy to embed the commands into your text: you just type them in like regular BEX words. BEX doesn't recognize them as commands, and sends them through to your printer. Once the printer gets them it starts doing the fancy stuff.

However, since BEX doesn't recognized the embedded printer control characters as commands, there are some drawbacks. The formatter counts these embedded commands as characters on the line: it doesn't know your printer won't print them. So, the printer control commands take up space on the line; the amount of space depends on the length of the command. When the commands are more than a couple characters, the lines containing the commands are noticeably shorter. You cannot solve this problem with a $$p# command; BEX formats each line before the printer gets it, so if you position the next word back over the control command, the command is erased and the printer never receives it. One solution for this is to use the $$eX command, disussed in the next Part.

Another drawback is that some printers default to normal print after every <CR>, which is usually at the end of every line. Every time characters printed with the control command are broken between lines, the characters on the second line revert back to normal. Also in this case BEX cannot help you.

Caution! Printer control commands are treated like spaces in the Grade 2 translator. Remove them from your text before you translate. If you don't remove them from your text, they may cause translation errors.

When your printer control commands include characters such as <ASCII 30>, <ASCII 31>, <control-S> or <control-T>, your printer will not receive them if $$ss or $$sd is active. When you need to deactivate the commands in order to send a control command to your printer, you must reset the formatter to default with $$d.

Part 5: Using a Specific Printer

When you define a printer in your configuration, you have seven printer classes to choose from. When you declare your printer as a *specific printer*, you can make use of 11 more format commands. These format commands control boldface, superscripts, subscripts, and other tasks. The essential difference between generic and specific printers is that BEX can command a printer configured as *specific* to execute these special tasks.

In all, there are five possible tasks. The various tasks are listed below. Your printer may not support all five, so you still have to refer to your printer manual. For example, the original Apple ImageWriter does not support subscripts and superscripts. BEX ignores any request for subscripts or superscripts sent to an original ImageWriter.

How does BEX know what codes to send? When you define a printer as *specific* in your configuration, BEX reads the control code information for your specific printer from the `PRINTERS` chapter on the Boot side into your configuration chapter. This information is read into your computer's memory when you boot BEX. Then, when the formatter encounters any of the 11 $$eX commands in your text, it reads the specific control code from memory, and

executes it. It is possible to alter the PRINTERS chapter to suit your taste. For example, while BEX defines $$e0 as 10 pitch for any specific printer, you could modify the table to make $$e0 mean headline mode. Altering the PRINTERS chapter is discussed in Part 6.

- ***Printer Features Supported***

You enter $$eX format commands in your text; BEX transforms into the appropriate printer control codes. The commands control the following features:

- 1. Printer-specific underlining (sending the printer its own underlining commands)
- 2. Pitch Change: 10 characters per inch (cpi); 12 cpi; 17 cpi
- 3. Bold letters
- 4. Superscript characters
- 5. Subscript characters

Not every printer has the ability to execute every one of these features. When the printer is not capable of executing a format command, then BEX doesn't send out any codes to the printer.

In contrast with putting your printer's own *escape codes* right in your text, using a $$eX command accomplishes the task very cleanly. In particular, the code that your printer gets from a $$eX command may contain several control characters, but the $$eX command does not increase the formatter's character count for the line of output. Thus the formatter considers only the characters of text when it decides where to break lines.

Also, any $$eX command in your text gets appropriate handling no matter what kind of printing device you are using. They do nit affect printing to the screen, voice and video review mode, etc. The same $$eX command has the appropriate result with whichever specific printer you use (as long as the particular task works on that specific printer). So you get the advantages of exotic printer codes without having to change your text for different printing devices. What's more you do not even have to know the printer's escape codes, except for deciding how to declare your printer in your configuration.

The $$eX commands do *not* work in running headers or footers. When you need bold type in a running header or footer, you have to include the appropriate escape sequence in your text, and not a $$eX command.

For a list of printers supported by BEX, see Section 4 of the Interface Guide. Part 6 tells how to add a new printer to the list of specific printers by modifying the PRINTERS chapter.

- ***Configuring a Specific Printer***

There is room in a configuration for only one list of escape codes. This means that you cannot configure one printer to be an Apple ImageWriter and another printer to be a Diablo 630. You can set up two different configurations, one for the ImageWriter and a second configuration for the Diablo.

However, you *can* configure two different printers in one configuration when one of the printers is configured as a generic printer. For example, configuring a Diablo as a specific printer plus an ImageWriter as a generic printer is okay in one configuration.

When you want to configure a specific printer, answer S when BEX prompts for a printer class. You are then asked `Is this a dot matrix printer?` Answer N for a daisy wheel printer. When BEX prompts `Enter printer code:` press <CR> for your choices. In Part 6 we explain how you can add a specific printer in the PRINTERS chapter. When you do, its number is automatically added to the list of specific printer codes.

- ***When Your Printer is not Listed***

There are hundreds of printers on the market. Obviously, this list only covers a small number of printers. If your printer is not on this list, it is probably compatible with one of the listed printers. Check your printer manual. You may find an explicit statement of compatibility or printer emulation. When the manual says a printer emulates an FX-80, for example, then configure it as an FX-80.

When your printer manual is lacking any statement of compatibility, it still may be compatible. You will have to do some research in your printer manual. First, look up the control codes for underlining and bold face printing in your printer manual. Next, boot up BEX and use option P - Printer control code display. Choose the type of printer, dot matrix or letter quality. As BEX lists each printer and its control codes, compare the codes to those of your own printer. When you find one that has codes that match perfectly with those of your printer, make note of that printer. Configure your printer as a class S - Specific printer, using that name. Everything should work well. When no listed printer is completely compatible, see Part 6: Adding your Printer to the List.

- ***Specific Printer Commands***

Eleven BEX $$ commands begin with $$e. These commands signal BEX to send out the printer control codes from the PRINTERS table. The letter *e* is a mnemonic for the fact that these commands usually generate *escape codes* for your printer. The letter *X* represents the number or letter for each specific command. All $$eX commands are suppressed when sent to any printer class except printers configured as a class S - Specific printer. For commands represented by letters, BEX uses the next letter in the alphabet for the "off" command. For example, $$eb is turn on boldface, and $$ec is turn off boldface.

Changing characters per inch

Only the following three values for the number of characters per inch (cpi) are supported:

- $$e0 - Pica (10 characters per inch)
- $$e2 - Elite (12 characters per inch)
- $$e7 - Condensed (17 characters per inch)
- Boldface
- $$eb - Boldface begin
- $$ec - Boldface end: *c* follows *b*
- Superscripts and subscripts
- $$ee - Elevate: start printing superscript
- $$ef - Stop printing superscript: *f* follows *e*

- $$el - Lower: start printing subscript
- $$em - Stop printing subscript: *m* follows *l*
- Escape code underlining
- $$eu - Underline begin
- $$ev - Underline end: *v* follows *u*

These commands behave very differently from the conventional*and* commands. The Grade 2 translator cannot use their appearance to place italics signs, and they are not affected by the $$sp special punctuation mode mentioned previously. In general, use*and* where ever possible.

However, two situations call for using the escape code underlining: when your printer does not interpret control-H to mean "back up one character" (some dot-matrix printers fall in this class) or if you're printing a form with many characters underlined and desire speedier output.

Printer commands and the touching token

Many applications of subscript and superscript will require using the touching token. Here's an example using subscripts:
H<control-T> $$el 2<control-T> $$em<control-T>O
(in words, that's uppercase *H,* subscript, the digit 2, stop subscript, uppercase *O;* otherwise known as the formula for water).

Part 6: Adding Your Printer to the List

Option P - Printer control code display on the Starting Menu prints the contents of the PRINTERS chapter to the HI-RES screen. For clearer voice output BEX spells each entry, using the arrowing vocabulary. BEX prints the table entries in order. Press <space> for the next printer; press <ESC> to return to the Starting Menu. When your printer does not match any of these combinations, do not panic. You can add to the table in BEX that contains the specific printer control codes.

The chapter called PRINTERS on the Boot side of your BEX disk contains the escape codes for the specific printers. You can inspect and add to these lists. To get a list of the escape commands for supported printers, use option P -

Printer control code display on the Starting Menu. This prints the contents of chapter PRINTERS:
Starting: P
This option displays the control
sequences for different brand name
printers.
Display codes for Dot Matrix?
Answer Y to this prompt only when you have a dot matrix printer. When you have a daisy wheel printer, answer N to this prompt. For example, answer Y when you have an ImageWriter; you answer N when you have a Diablo 630. Consult your printer manual if you are unsure about your type of printer.
After you answer the dot matrix question, BEX displays the list of control codes for each printer. If any command is not available for that printer, BEX tells you. You press any key to move to the next printer listed.

To add your printer to the PRINTERS chapter, first make a backup copy of PRINTERS. Treat chapter PRINTERS like any other BEX chapter. Use the Editor to change the PRINTERS chapter. Page one is for letter quality printers, page two is for dot matrix printers. Page three is for the Apple LaserWriter Postscript Driver. If you're interested in how this works, write us for details.

Each item in the table is separated by a delete character. The end of the table is indicated by two backslashes. Each printer name starts with a backslash. To add a new printer to the list, delete the last backslash, then type in the printer codes. Type the codes in the order listed below. After each item, enter <DEL>. When the printer does not support a feature, enter <DEL> alone for that item.

To make it easier, make a block copy of one printer's list as an outline for your own, and clipboard it onto the end of the list. When you do this, make sure you have altered each entry. End the page with <DEL> for the last entry, then two backslashes to signal the end of the list: `<DEL>\\`

Here are the fourteen items required for each printer:

- 1. Name of the printer
- 2. Backup character (usually <control-H>)
- 3. 10 cpi (characters per inch)

- 4. 12 cpi
- 5. 17 cpi
- 6. Proportional spacing
- 7. Underline start
- 8. Underline stop
- 9. Boldface start
- 10. Boldface stop
- 11. Superscript start
- 12. Superscript stop
- 13. Subscript start
- 14. Subscript stop

There are 14 delete characters in each printer's entry. Don't forget to finish the page with <DEL> and double backslash. Once you are sure you have the right codes in the table formatted properly, set up a new configuration referencing your specific printer.

You have two tests to see if you've entered the codes correctly: The first is to see if option P - Printer control code display on the Starting Menu lists your printer. This means BEX can use the table, because you've entered the correct number of <DEL>s and backslashes. The second test is in using your printer. Only by using your printer will you know for sure that you have entered the right codes.

As we mentioned at the start, BEX uses the entries from this table when it executes all $$eX commands. For example, the third entry, 10 characters per inch, defines what BEX sends to your printer as it executes the $$e0 command. If you want to, you can enter the escape codes that start double wide printing in place of the codes for ten characters per inch. When you do, $$e0 turns on double wide printing for that printer.

6 Contextual Replace

By now you're familiar with option R - Replace characters on the Main Menu. You know that you can use Replace characters to dramatically modify text. Contextual Replace adds two more tools to the Replace characters workbench: *pattern strings* and *on* and *off strings.* A contextual transformation chapter can use either the pattern string tool, the on or off string tool, or both.

Part 1: Overview of the Tools

In basic Replace, each transformation rule pairs a *find string* with a *change to* string. In contextual Replace, each transformation rule has three parts: the find string, the *pattern string,* and the change to string. Pattern strings consist of special characters, (called *pattern codes*), that provide you with wild card functions. One pattern code means "find this letter whether it's upper- or lowercase." Another pattern code means "find any digit." We explore all thirty-four pattern codes in great detail in Parts 3 and 4. You can combine the various pattern codes to do some pretty nifty things.

Suppose you want to change lowercase letters to uppercase letters. With basic Replace, all your finding and changing must be explicit; you must write 26 transformation rules--like the LCUC transformation chapter on the BEXtras disk. Find *a* and change to *A;* find *b* and change to But, and so on through Zebra. Contextual Replace's pattern strings enable you to write a single transformation rule that finds every lowercase letter and changes it to its uppercase equivalent.

In contextual Replace, the find string alone does not specify what's found. It's the *combination* of the find string and the pattern string that defines what's found and where the change to string is placed in the target chapter.

On and *off strings* allow you to selectively Replace material within a chapter. You decide on a group of characters that serve as a "green light" or beginning point to start replacing. Once the program encounters this *on string,* it starts executing all the transformation rules you've supplied. You can also define a string of characters that serves as a "red light" or stopping point; this *off string* halts execution of the transformation rules. When the program encounters the off string, it doesn't stop altogether. Rather, Replace characters continues to search through the text for another *on string* that would reactivate execution of the transformation rules. You may define the on and off strings as anything you wish; they could be characters naturally part of the data you're transforming, or they could be characters that you insert for the specific purpose of switching replacement off and on.

As a quick example, you could define BEX's center-and-underline command, $$h as your on string, and BEX's paragraph ($p) indicator as your off string. Combining these on and off strings with the single case-changing transformation rule mentioned above, you can use Replace characters to make all your headings uppercase.

The transformation chapter that accomplishes the feat in the above example is just 22 characters long. But this brevity has its costs: Contextual Replace's pattern codes are quite cryptic. In fact, contextual Replace is almost a programming language. Like any language, you need to learn the vocabulary and the syntax to become a fluent user. And as with all language acquisition, learning about contextual Replace is not a linear process. You may find yourself reading and rereading this Section, and wondering if you will ever understand it. Then one day--whoosh!--it all makes sense.

Allow enough time for the learning process

Unlike some of the other material in the BEX Dox, you must read this Section front to back to make the most of the information provided here. Here's why: As you begin learning any language, you only know a few words and a few ways to put the words together. It's difficult to be eloquent until you have a range of vocabulary and syntax under your belt. In the following pages, we provide many elementary samples to assist you in understanding contextual Replace's vocabulary and syntax. Don't let the simplicity of these samples mislead you--once you have a full understanding of contextual Replace, you are indeed a Master of BEX.

Part 2: The Elements of Contextual Transformation Chapters

In User Level Section 8, we provided a sample of typing changes directly that illuminates how basic transformation chapters are structured. The same process should provide insight into how contextual transformation chapters are put together. But before you can start typing contextual changes directly, you need to learn a little bit about on and off strings and pattern codes.

As mentioned briefly in Part 1, on and off strings allow you to activate and deactivate the execution of transformation rules within a chapter. With basic Replace, you define a unique character to serve as a *terminator.* The terminator defines the end of the find and change to strings. This is also true in contextual Replace. Additionally, you *may* define two unique strings that serve as the signals to start and stop replacement. You don't *have to* define on and off strings: When you enter no characters as on or off strings, then all transformation rules are executed for the entire chapter.

The on and off strings may be any length, from 1 to 80 characters. Generally, the on and off strings are different from each other. In any one transformation chapter, the characters you define as on and off strings cannot be replaced. (When you're a real hotshot you can break these rules--details in Part 5.)

As mentioned in Part 1, contextual transformation rules consist of three parts: find, pattern, and change to strings. Each pattern string is made up of pattern codes; each code has a specific meaning. Each character in the find string is paired with one pattern code in the pattern string. The combination of the find string character and the pattern code define what happens during replacement. An example of a pattern code is the lowercase letter it. When a find string character is paired with the lowercase x pattern code, it means *match exactly this find string character and remove it in the target chapter.* In other words, when you use this pattern code, your contextual transformation rules function exactly like basic transformation rules.

- ***Typing Contextual Changes Directly***

The best way to learn about the elements of contextual Replace is to just dive right in and try it. Here's the first task: In the `QUANDARY` chapter, change almost every appearance of the word *blind* to the words *vision impaired.* Don't change *blind* to vision impaired when the word appears in a heading. Here's how you type those change directly:
Main: R
Replace
Chapter: QUANDARY <CR>
Chapter: <CR>
Target chapter: QUANDARY-C <CR>

Use transformation chapter: <CR>
Enter terminator: #
So far, this dialog is familiar. Press R, supply source and target chapter names, press <CR> at the `Use transformation chapter:` prompt to type changes directly, and establish which character serves as the terminator, in this case, the number sign.

When you enter your single terminator character at the first `Find:` prompt, you signal BEX to begin contextual Replace. Here's how it looks:
Find: #
Contextual replace
On string: <space>$p <space>#
Off string: $$p-1 $$c#
BEX responds with `Contextual Replace` and prompts you to define your on and off strings. We don't want to change the word *blind* to vision impaired when it appears in a heading. The headings in the QUANDARY chapter all begin with $$c and end with a paragraph ($p) indicator. We want to turn replacement *off* after the $$c and turn it back *on* at the start of the following paragraph. That's why we define the on string as ($p) and the off string as $$c. Here's the rest of the dialogue:
Find: blind#
Pattern: xxxxx#
Change to: vision impaired#
Find: #
Pattern: #
Continue? Y <CR>
Chapter QUANDARY done
Replaced 9 times
Save transformation chapter: QT <CR>

Once the on and off strings are defined, you begin to create transformation rules. The transformation rule begins with a find string. In this case, we're finding the word *blind.* Then comes the pattern string. Each character in the find string is paired with the pattern code character in the same relative position. The first find string character pairs with the first pattern code; the second find string character pairs with the second pattern code, and so forth. In this sample, each letter in the word *blind* is paired with the pattern code lowercase it. There are five letters in the word *blind,* so there are five lowercase x pattern codes.

The change to string functions exactly as it does in basic Replace. It defines what the find string is replaced by; in this case, the words *vision impaired.* The *combination* of the find character and the pattern code define what BEX finds and changes in your chapter.

As always, the change to string always ends with the terminator. When you press your single terminator character at both the `Find:` and `Pattern:` prompts, you signal the end of the list of rules. Our sample has only one transformation rule, so we press the number sign terminator at the `Find:` and `Pattern:` prompts. As with basic Replace, BEX prompts `Continue? Y` and gives you one last chance to chicken out or switch disks. Press <CR> here to start executing the rules.

The structure of contextual Transformation chapters

As with basic Replace, you have an opportunity to save the list of transformation rules in a transformation chapter. In the sample, we save the rules in a chapter named QT. When you edit the QT chapter, you find all the characters you typed between the `Use transformation chapter:` and `Continue? Y` prompts. It looks like this:

$p #$$c#blind#xxxxx#vision impaired###
All contextual transformation chapters begin with two terminators. Between the second and third terminator are the characters that serve as the *on string.* The *off string* consists of the characters between the third and fourth terminator. After these four terminators, the list of transformation rules starts.

Each contextual transformation rule consists of the *find string,* a terminator, the *pattern string,* a terminator, the *change to* string, and one more terminator. The list of rules in a contextual transformation chapter always ends with three terminators in a row. The total number of terminators in a contextual transformation chapter is always divisible by three.

When your pattern string contains all lowercase x pattern codes, then Replace only finds characters that exactly match the find string, removes them from the target chapter, and replaces them with the change to string. (Exactly what basic Replace does.) With the QT sample, we have only taken

advantage of one of contextual Replace's tools: the ability to switch replacement on and off within a chapter.

We now change gears and discuss pattern codes in greater detail. We return to the topic of on and off strings again in Part 5.

Part 3: Pattern Codes in Detail

So far, the only pattern code we've discussed is lowercase it. If that were the only pattern code, then contextual Replace wouldn't be much of an improvement over basic Replace. But lowercase x is just one of 34 pattern codes that together provide you with a flexible, if cryptic, way of manipulating text.

Three pattern code groups

Every pattern string must contain some number of pattern codes. The pattern codes divide into three groups: *departing* codes, *boundary* codes, and *specials.* How the four specials work is a little complicated--we defer a complete explanation until Part 4.

The fifteen departing pattern codes and fifteen boundary pattern codes are drawn from the same set of fifteen letters. When the letter is *lowercase,* it's a departing pattern code. When the letter is *uppercase,* it's a boundary pattern code. Some of the fifteen letters are mnemonic, but we ran out of clever names halfway through. Part 8 and the Thick Reference Card provide an alphabetical listing of all pattern codes, so don't try too hard to memorize them all.

In the sample in Part 2, the QT contextual transformation chapter pairs a *departing* pattern code with every character in the find string: the lowercase letter it. The find string characters are removed from the target chapter.

When you pair a find string character with a *departing* pattern code, then the character in the source chapter that satisfies the combination of find and pattern strings is removed from the target chapter. The departing pattern codes show BEX where to place the change to string in the target chapter.

When you pair a find string character with a *boundary* pattern code, then the character in the source chapter that satisfies the combination of find and pattern strings remains in the target chapter. The boundary pattern codes allow you to specify a *context* for replacement--one of the reasons we call this option *contextual* Replace.

Caution! The only significant spaces in this Section are those indicated by <space>. Any other spaces are there for ease of reading.

Here's a single transformation rule that illustrates the difference between departing and boundary pattern codes. The task is to change the Roman numeral II to the Arabic digit 2 whenever the text discusses literary braille. The terminator is the number sign:
Find: Grade <space> II#
Pattern: XXXXXXxx#
Change to: 2#
The first six characters of the find string are paired with uppercase X boundary codes in the pattern string. The only time that Roman numeral II is replaced with Arabic digit 2 is when the six characters preceding the Roman numeral are exactly uppercase *G,* lowercase *r,* lowercase *a,* lowercase *d,* lowercase *e,* <space>. Clever readers will note that a basic Replace rule that finds `grade 2` and changes it to `Grade II` accomplishes the same task. But stay tuned! We return to this issue again shortly.

The contiguous departers requirement

One rule governs where boundary pattern codes may appear in the pattern string. Boundary codes cannot interrupt a string of departing pattern codes. Boundary codes can appear as the initial characters in the pattern string, as the final characters in the pattern string, or both initial and final. But they cannot appear in the middle of a string of departing pattern codes. Departing pattern codes must be *contiguous.* That is, departing pattern codes must touch in unbroken sequence. The departing pattern codes define where BEX places the change to string in the target chapter. If the departing pattern codes were interrupted by boundary codes, BEX wouldn't know where the change to string should go. The "II to 2" rule above shows initial boundary codes; the other possibilities are shown in some of the many samples that lie ahead.

Find and pattern strings are the same length

Each pattern code is paired with one character in the find string. Therefore, the find strings and the pattern strings must be the same number of characters. When you type contextual changes directly, BEX refuses to accept a pattern string that's shorter or longer than the find string.

With this basic information under your belt, you're ready for the guided tour through the pattern codes. In the rest of this Part, we examine each pattern code in detail. Some pattern codes are pretty obvious; we just describe what they do. Many other pattern codes are more subtle. To make their function clear, we provide samples of where you could *use* the pattern code under discussion. Some of the samples show a lowercase *departing* pattern code, others show an uppercase *boundary* pattern code. Remember, each of the fifteen pattern code letters can be used either way.

- ***Pattern code B: Blank Space***

This pattern code stands for just one character, the space. (Think of the letter *b* in the word *blank*.) It's actually a redundant pattern code, since you can also specify the space by pairing a space in your find string with the exact pattern code X or it. We included this pattern code to help make your transformation rules a little more legible, as you see in subsequent samples.

- ***Pattern code W: Total Wild Card***

The pattern code letter W means every possible character you can have in a BEX chapter. (Think of the letter *w* in the word *wild*.) Technically, this means every ASCII character from zero to 127; this includes all control characters, the space character, all the digits and letters, all punctuation and symbols. Because the W pattern code stands for every possible character, it really does not matter which find string character you pair it to.

Analyze the data before you write the rules

Before you can write successful transformation rules, you must analyze the patterns inherent in the data you're changing. Both contextual and basic Replace execute your transformation rules in order, one BEX page at a time. The program compares the first rule with each and every character. When it

finds a match, then the change to string characters are inserted in the text. Once all the characters are compared with the first transformation rule, BEX repeats the process with the second transformation rule. It's crucial to consider how the initial rules in a transformation chapter may change the patterns inherent in the data you're replacing. We return to this topic again after we provide a sample using the lowercase w wildcard.

Sample: deleting BEX format commands

Your BEXtras disk contains a chapter named `RESUME` that's stuffed full of various $$ commands. The task here is to delete every BEX $$ format command from this chapter. This is possible because BEX format commands follow a pattern. They all begin with two dollar signs. They all end with a space. (We assume here that the text is formatted with BEX's new-line ($l) and paragraph ($p) indicators. Later on in this Part, we discuss how to cope when you format text with hard <CR> s.)

BEX format commands can vary in length. In between the two dollar signs and the space, there are some number of lowercase letters and frequently some digits as well. Some format commands contain punctuation symbols, like the plus sign, minus sign, or asterisk. We write six transformation rules to find and delete format commands from three to seven character long.

```
Main: R
Replace
Chapter: RESUME <CR>
Chapter: <CR>
Target Chapter: 1RESUME-NO$ <CR>
Use transformation chapter: <CR>
Enter terminator: #
Find: #
Contextual replace
On string: #
Off string: #
Find: <space> $$<space>#
Pattern: BxxB#
Change to: -#
Find: $$#
Pattern: xx#
Change to: <space> $$#
Find: $$c <space>#
```

Pattern: xxwB#
Change to: #
Find: $$p5<space>#
Pattern: xxwwB#
Change to: #
Find: $$ml4<space>#
Pattern: xxwwwB#
Change to: #
Find:<space>#
Pattern: xxwwwwB#
Change to: #
Find: $$ml+24<space>#
Pattern: xxwwwwwB#
Change to: #
Find: #
Pattern: #
Continue? Y <CR>
Starting to replace
Replaced 119 times
Save transformation chapter: STRIP DOLLAR <CR>

This sample shows that a contextual transformation chapter may contain more than one pattern code. In fact, a contextual transformation chapter could contain many combinations of all 34 departing and boundary pattern codes.

The first rule's find string is BEX's move-to-the-next tab command, were $$). This is the only $$ command that doesn't follow the pattern described above. We replace the tab with a hyphen, since we still want to have some characters separating the text that was formatted with tabs.

The second rule ensures that all subsequent $$ commands follow a regular pattern, that is, they all start with two dollar signs and end with a space. Although we urge you to precede and follow every $$ command with a space, it's also possible to jam a series of $$ commands together. All the other rules in this transformation chapter depend on a boundary space after the $$ command. As it happens, the RESUME chapter contains this text:
... long-range planning of the firm.<space> $$mr-8<space> $p <space>
NORTH FARM COOPERATIVE ...

If the second rule did not insert a space before every $$, then the could not be changed, since it does not end with a space.

The next five rules specify every possible format command, starting with three-character-long commands and working up to seven-character-long commands. The third rule's find string is one possible format command that's three characters long: $$c. The pattern string begins with two lowercase x pattern codes that define exactly two dollar signs as departing characters. While the next departing pattern code, lowercase w, is paired with the specific character *c,* because it's a total wild card, this rule also matches $$b, $$d, $$h, $$r, and $$z. The last pattern code is a boundary uppercase B, standing for a space. (Uppercase X means the same thing; the B code here improves readability.) Since we enter the number sign terminator at the `Change to:` prompt, the change to string is empty. Since the initial dollar signs and the next character are replaced by nothing, they're deleted from the target chapter.

The fourth rule's find string shows the $$p5 format command. Since the *p5* characters are paired with two lowercase w pattern codes, this rule also matches $$a3, , $$ub, and any other format command that's four characters long. The fifth, sixth, and last rules use the same technique to find and delete format commands from five to seven characters in length.

When you edit the RESUME-NO. chapter, you find many extra spaces. These spaces separated $$ commands in the original RESUME chapter, and were "thrown away" when BEX prints. You could use the `SP2` transformation chapter on your BEXtras disk to delete all the extra spaces. (In Part 7, we explain how SP2 works.) Once you add more pattern codes to your repertory, it's possible to rewrite these rules to not create those extra spaces--we provide an example of this later on in this Part.

Reversing the order would delete characters

The order of the transformation rules in the STRIP DOLLAR chapter is crucial. If you reversed the order of the rules, searching first for seven-character format commands, next for six-character, and so on, you could delete portions of your text in addition to the $$ commands. Suppose the RESUME chapter contained the following characters:
...<space> $l <space> $$mr8 <space> A <space> mail-order ...
If the third transformation rule was:

Find: $$ml+24<space>#
Pattern: xxwwwwwB#
Change to: #
then the line of text would get mangled to:
...<space>$l <space><space> mail-order <space> firm ...

Here's why: The departing pattern code w matches every possible character. The transformation rule really says, delete two dollar signs and the next five characters, as long as the space boundary exists. It just so happens that $$mr8<space>A<space> satisfies this rule. These are five characters between the initial two dollar signs and for <space> after the article *.A.*

Because the STRIP DOLLAR transformation chapter begins with short $$ commands and finishes with long ones, it avoids this sort of problem. The rule that matches five-character $$ commands *precedes* the rule that matches seven-character $$ commands. Since Replace characters executes rules in order, the five-character command $$mr8 is matched and deleted *before* Replace starts looking for seven-character format commands. By the time Replace characters tries to match the seven-character rule, the `$$mr8 <space> A <space>` text no longer exists.

- ***Pattern code I: Ignoring Capitalization***

You pair the pattern code I with a letter in your find string. Pattern code I means: find exactly this letter in two situations, when the find string letter is uppercase *and* when it's lowercase. (Think of the letter *i* in the word *ignore*.) When the pattern code is uppercase I, then its find string partner is a boundary, and remains in the target chapter. When the pattern code is lowercase i, then its find string partner departs. Here's how we modify the "II to 2" rule from the QT transformation chapter (shown in Part 2) to make it more general (the terminator is number sign again):
Find: Grade <space> II#
Pattern: IIIIIBxx#
Change to: 2#
This rule finds and changes the Roman numeral II in a variety of contexts. Because the boundary code I ignores capitalization, *Grade II, grade II,* and *GRADE II* all satisfy the combination of find and pattern strings.

- ***Pattern codes S, U, and L: Generic Letters***

These three pattern codes focus on the letters of the alphabet. The pattern code S stands for *any* letter of the alphabet, as long as it's lowercase. (Think of the letter *s* in the word *small*.) The pattern code U is the opposite of S; U stands for any letter of the alphabet that's uppercase. (Think of the letter *u* in the word *uppercase*.) Finally, the pattern code L stands for 52 characters: any letter of the alphabet, regardless of capitalization. (Think of the letter *l* in the word *letter*.) The following sample illustrates what a time-saver the S, U, and L pattern codes can be.

Sample: Inserting underlining for single letters

The task here is one we actually faced while revising this edition of the BEX Dox. In the earlier edition, when we were discussing an individual letter, we would enclose that letter in double quotes. In this edition, we italicize a single letter. We didn't want to make all those changes manually! The following sample transformation rules delete the quotes from around a single letter, and insert BEX's underlining commands,*and*.

The number sign serves as the terminator once more. The first rule changes the opening double quote:
Find: <space> "A" <space>#
Pattern: BxLXB#
Change to: $$p-1 <space>#
We're assuming that a quoted single letter, together with any touching punctuation, is preceded and followed by a space. The find string shows the letter *A*, but since it's paired with the boundary code L, this rule actually applies to every possible letter, regardless of capitalization. With basic Replace, we would have had to write 52 rules to accomplish the same task. It's important to include both the opening and closing quotes, because we only want to change double quotes around *single* letters. Only one pattern code is departing; the double quote is removed from the target chapter. The change to string consists of the start underlining command.

This rule illustrates the contiguous departers requirement. Because departing pattern codes must touch, a pattern string of `BxLxB` is not allowed. When

you want to modify text on both sides of a given group of characters, you have to work on one side first, and then the other.

Now that we've dealt with the opening double quote, the second rule handles the closing double quote.
Find: $$ub<space>A"<space>#
Pattern: XXXXBLxB#
Change to: <space>$$uf#
Replace characters always executes the transformation rules in the order you enter them. Therefore, after the first rule, the emphasized letters no longer have opening quotes--they have the begin underlining format command, $$ub. The find string for the second rule reflects this change. The pattern string again contains a single departing code, paired to the closing double quote. The change to string does not need to include a space *after* the $$uf, because the space after the double quote is paired with a boundary pattern code, B, which is not changed in the target chapter. This space is still present in the target chapter, where it serves as the space required after every BEX format command.

These two transformation rules accomplish the task at hand for many occurrences of quoted letters. The only exceptions are when punctuation touch the quoted letter. One could write a series of rules like this:
Find: <space>"A,"<space>#
Pattern: BxLXXB#
Change to:<space>#
making one rule for every possible punctuation that touches the quote marks. In addition to the comma shown, you would need rules for the period, exclamation point, question mark, colon, left and right parentheses, left and right brackets, semicolon, and several more. But (as you're probably not surprised to discover at this point) contextual Replace has one pattern code for this situation.

- ***Pattern code P: punctuation and symbols***

The pattern code P stands for punctuation and symbols. (Think of the letter *p* in the word *punctuation*.) Its definition is actually a process of elimination: P means anything that's not a letter, not a digit, not a control character, and not a space. Note that many of the symbols on this list are not what your English

teacher taught you as "punctuation." For the record, a complete list of the characters that match the p pattern code appears in Part 8.

Sample: Inserting underlining for single letters, continued

Returning to our underlining letters transformation chapter, the P pattern code is used to match various combinations of punctuation touching quotations. The next two rules place the begin and finish underlining when a symbol of enclosure--parentheses, brackets, etc.--precedes the opening quote:
Find: <space>("A"<space>#
Pattern: BPxLXB#
Change to: <control-T><space>#
Find: (<control-T>$$ub<space>A"<space>#
Pattern: PXXXXXBLxB#
Change to: <space>$$uf#
The <control-T> *touching token* is discussed fully in Section 5, Part 1; it can replace the initial or final space in a BEX format command. Because we only want to underline the letter, and not the punctuation that precedes it, we need to begin underlining after the initial punctuation. But if we inserted <space> $$ub <space> there would be one space between the initial punctuation and the letter. We replace the initial space with <control-T> so we can start underlining within a BEX word. BEX throws away the final space in the underlining command when it's executed.

The next two rules cope with punctuation that appears to the right of the closing quote:
Find: <space>"A".<space>#
Pattern: BxLXPB#
Change to: $$p-1 $$ub <space>#
Find: <space>$$ub<space> A".<space>#
Pattern: BXXXXBLxPB#
Change to: <control-T>$$uf<control-T>#
The first <control-T> ensures that there's no space between the letter and final punctuation. The second <control-T> signals the end of the $$ command, so that the translator turns on again and correctly translates the punctuation. Remember, although the find string shows a period after the closing quote, when paired with the boundary pattern code P, it matches a semicolon, right parentheses, or any of the other characters from the long list above.

Two more rules cope with the remaining possibility, when the punctuation appears between the quoted letter and the closing quote:
Find: <space>"A,"<space>#
Pattern: BxLPXB#
Change to: $$ub<space>#
Find: <space>$$ub<space>A."<space>#
Pattern: BXXXXBLPxB#
Change to: <space>#

We don't bother with <control-T> s here, because we use the $$sp command so BEX suppresses underlining of final periods, commas, semicolons, and colons. A more elegant transformation chapter that accomplishes the same task with fewer rules is discussed in Part 6.

- ***Pattern code D: Delimiting BEX "Words"***

The D pattern code stands for two characters: either <CR> or <space>. (Think of the letter *d* in the word *delimiter*.) A <CR> or space is how BEX defines a "word" in the Editor. So far, our examples have assumed you format your text with BEX's new-line ($l) and paragraph ($p) indicators. When you use hard <CR> s to format your text, then the B blank pattern code does not sufficiently define a word. For example, the last word on a line begins with a space but ends with <CR>. To make the STRIP DOLLAR transformation chapter work correctly when your text contains hard <CR> s, substitute the D pattern code for the B pattern code.

- ***Pattern code E: Everything Except Space or <CR>***

The E pattern code is the opposite of the D pattern code. E stands for every character except two: <space> and <CR>. (Think of the letter *e* in the words *everything but*.) When this code is departing, lowercase e, it's a succinct way to delete words.

Sample: Deleting BEX format commands, revisited

When we introduced the wild card pattern code W, we showed six rules that deleted BEX format commands. Now that we have the D pattern code in our

repertory, we can write transformation rules that find BEX format commands ending with both <space> and <CR>. And now that the E pattern code is at our disposal, we can delete all BEX format commands with just four rules.
Find: <space> $$ <space>#
Pattern: BxxB#
Change to: #
Find: $$p-1 $$x#
Pattern: XXe#
Change to: #
Find: $$p-1 $$<space>#
Pattern: xxb#
Change to: #
Find: $$p-1 $$<CR>
Pattern: xxD
Change to: #
The first rule finds the move-to-the-next tab command, inserting a hyphen to separate what was separated with tabs. The second rule finds any character following two dollar signs *except* a <space> or <CR>. BEX repeatedly executes this rule, deleting one character at a time from all format commands. Once all the characters between the two dollar signs and the delimiter are gone, the third and fourth rules clean up what's left. The third rule deletes what's left when the format command was *not* at the end of a line. The fourth rule deletes the two dollars signs left when the format command ended with <CR>, but leaves the <CR> itself. Try these rules on the RESUME chapter and see what happens!

- ***Pattern code Q: Defining Real Words***

The Q pattern code is a combination of the D and the P pattern codes. It stands for space, <CR>, or any punctuation or symbol. Although BEX treats the punctuation touching a word as part of the word for Editor cursor movement, many times you don't want that punctuation considered as part of the word when replacing.

Sample: Expanding keyboard shortcuts

As we promised in User Level Section 8, contextual Replace makes it much easier to develop keyboard shortcuts for your data entry. Suppose you type

the three characters *s-b* to stand for the word *SlotBuster* and the two letters *vi* to stand for the two words *visually impaired.* The following basic Replace rules would cause problems:
Find: s-b#
Change to: SlotBuster#
Find: vi#
Change to: visually <space> impaired#
When your original text has a sentence like *The villainous has-been twirled his mustache,* then the above rules create the nonsensical *the visually impairedllainous haSlotBustereen twirled his mustache.*

Aha! you think, I'll just modify the rules to include a space before and after the shortcuts, like this:
Find: <space> s-b <space>#
Change to: <space> SlotBuster <space>#
Find: <space> vi <space>#
Change to: <space> visually <space> impaired <space>#
but then neither keyboard shortcut is matched in this sentence: *The "s-b" card is useful to people who are vi.*

The contextual Replace pattern code Q comes to the rescue, as follows:
Find: (Vi)#
Pattern: QIxQ#
Change to: ision impaired#
Find: (s-b)#
Pattern: QxxxQ
Change to: SlotBuster#
Although the find strings show left and right parentheses, when paired with the Q pattern code, these rules find and change *vi* and *s-b* in a host of different contexts, whenever the shortcut letters appear as a word. By pairing the initial letter *v* with the I boundary code, we match the shortcut *Vi* at the start of a sentence. The first letter of the shortcut remains in the target chapter, with the same case as it had in the source chapter. *Villainous* and *has-been* don't satisfy the Q pattern code, so you don't end up with nonsense. However, if your text contains Roman numeral six, you're in trouble. As always, it's important to know your data before you start writing rules.

- ***Pattern codes N and A: Numbers***

The N pattern code stands for the digits zero through nine. (Think of the *n* in the word *numeral*.) The A pattern code stands for 62 characters; the ten digits plus any letter, lowercase or uppercase. (Think of the letter *a* in the word alpha-numeric.) For those who like formulas, pattern code L plus pattern code N equals pattern code A.

- ***Pattern code O: Wild Card Minus One***

The pattern code letter O is a very handy code indeed. It stands for every possible character except one--the find string character it's paired with. (Think of the letter *o* in the words *other than*.) When pattern code O is paired with the dollar sign, then it matches every character *except* the dollar sign.

The sample task here is again drawn from our experience at RDC. For formatting our large print output, we use JustText, an embedded command typesetting program for the Macintosh Plus. All its commands begin with left brace and end with right brace. The JustText commands vary greatly in length; some are five characters long, while others are twenty characters long. We usually prepare our text totally with BEX, using contextual Replace to put the JustText commands where they should go. Occasionally, however, we do data entry directly in JustText. When we bring this data back to the Apple, most of the JustText commands are irrelevant.

The transformation chapter that strips out the JustText commands begins with several rules that change the few meaningful JustText commands back to their BEX equivalents--we won't bother to show you those. It's the last two rules in the transformation chapter that show off the power of the o pattern code:
Find: {}#
Pattern: Xo#
Change to: #
Find: {}#
Pattern: xx#
Change to: #
The first rule matches the beginning of a JustText command. The initial left brace is paired with uppercase X, so it remains in the target chapter. The right brace, when paired with departing o, means: *delete any character except the right brace.* This rule is executed repeatedly, so it deletes each character in

the JustText command until all that's left are the initial and final braces. The second rule deletes this residue.

- ***Pattern code C: Control Characters***

The C pattern code stands for any control character--technically speaking, the ASCII values zero to 31 plus ASCII 127. (Think of the letter *c* in the word *control*.) In terms of characters you are likely to enter in a BEX chapter, the C code includes <ESC>, <CR>, <control-T>, <control-S>, <ASCII 30> (the discretionary line-break), <ASCII 31> (the discretionary hyphen), and <DEL>.

The C code can be very useful when you are working with files from other computers that arrive at your Apple full of printer control commands. Suppose you have a file like this; the only control character you want to preserve is <CR>. Assuming that all printer control commands are two characters long, you can reformat the text with these rules:
Find: <CR>#
Pattern: x#
Change to: <space>$l <space>#
Find: <ESC> O#
Pattern: cw#
Change to: #
Find: <space>$l <space>#
Pattern: xxxx#
Change to: <CR>#
These rules demonstrate another important technique for Replace characters, both basic and contextual: temporarily *protecting* a character from being changed. The first rule changes <CR> to the new-line ($l) indicator. The only control characters that remain are ones you don't want, so the second rule deletes them. The third rule reverses the action of the first rule, changing the new-line indicators back to hard <CR> s.

Summary

We've now defined the fifteen departing and boundary pattern codes. We've also provided samples for most of them. Now it's time to dive into the mysterious world of the *specials.*

Part 4: The Special Pattern Codes

The four *specials* "break the rules" we've established so far. But in return for this inconsistency, you get the ability to do even niftier things. The special pattern codes are the caret, uppercase Z, uppercase Y, and the current terminator. Let's look at these in reverse order.

- ### *The Pattern String Shortcut*

In the very first contextual Replace sample, the pattern string consisted of all lowercase x pattern codes. Just to make your life a little easier, there's a shorter way to indicate that you want an exact, departing match for every character in your find string. Instead of entering a lowercase x for every character in your find string, enter your single terminator character at the `Pattern:` prompt. The original rule looked like this:
Find: blind#
Pattern: xxxxx#
Change to: vision <space> impaired#
The next sample functions identically to this rule, using the pattern string shortcut:
Find: blind#
Pattern: #
Change to: vision <space> impaired#

- ### *Pattern codes Y and Z: Case-changing Specials*

The uppercase letters Y and Z are like boundary pattern codes with one difference. All the boundary codes result in no change in your target chapter. Y and Z create a subtle change in your target chapters. When the combination of a find string character and the pattern code Y or Z is satisfied, the find string character changes its case. Uppercase Y stands for any letter, just like the letter code L; but when the find string partner is a lowercase letter, then that letter is uppercase in your target chapter. Uppercase Z stands for any letter; when the find string partner is an uppercase letter, it becomes lowercase in your target chapter.

Caution! The Y and Z do not automatically *reverse* the case of the find string partner letter. Y forces an uppercase letter, and Z forces a lowercase letter. With the exception of their case-changing abilities, Y and Z follow the rules for boundary pattern codes.

- ***Pattern code caret: Insert Change To String Here***

Up until now, all the pattern strings we've shown contain at least one departing character. Sometimes, however, you don't *want* to remove any of the characters in your source chapter that satisfy the combination of find string and pattern string. The caret ^ character means: *insert the change to string characters at this point.* With this special pattern code, you can make changes without removing anything.

Now that the full truth is out, we can state the full requirements for pattern strings. A pattern string cannot consist solely of boundary pattern codes. A pattern string can contain any number of boundary pattern codes, plus *either* exactly one caret, or at least one departing character. When a pattern string does not contain the caret, then the number of characters in the pattern string is exactly the same as in the find string. When a pattern string contains a caret, then the pattern string is one character longer than the find string.

You cannot have more than one caret in any one pattern string, because the caret shows BEX where to insert the change to string. If you had two carets, BEX would not know where to put the new characters. Any one pattern string can not contain both a caret and any departing pattern codes. When you find yourself wanting to mix and match this way, you need to write two separate rules.

Sample: Inserting commas in long numbers

You've written a report with many large dollar amounts and other numbers in it, but you realize that you've left out all the commas between the hundreds and the thousands places. You can insert the comma using the caret code (the number sign is the terminator):
Find: 4321.#
Pattern: N^NNNQ#
Change to: ,#

This rule matches any four numbers, followed by punctuation or a delimiter, and inserts a comma between the first and second numbers. This takes care of a lot of cases. It doesn't matter whether the numbers are whole or followed by two decimal places or whether it's prec $$ by a dollar sign. However, this rule only works for numbers in the thousands: when your data contains millions and billions, you need to write more rules.

But at the end of your report, there's a list of phone numbers. You don't want commas between the fourth and fifth digits in those. You see that what makes these numbers unique is the hyphen between the third and fourth digits. The O pattern code specifies a match for every character *except* its find string partner--in this case, the hyphen:
Find: -4321.#
Pattern: ON^NNNQ#
Change to: ,#
That's better! Now the rule matches four digits followed by a space or delimiter, and inserts a comma between the first and second digits, *unless* the first number is preceded by a hyphen.

Sample: Uppercase to lowercase

You can make great use of the caret when the change to string is empty. The following single rule accomplishes the same task as the 106-character UCLC basic transformation chapter on your BEXtras disk.
Find: A#
Pattern: ^Z#
Change to: #
The find string contains the letter *A.* When paired with the boundary pattern code Z, this means find every letter and make it lowercase. A pattern string can not consist totally of boundary codes. We precede the Z with the caret, to mean *insert the change to characters here* then pull the sneaky trick of creating an empty change to string. No text is inserted; the target chapter contains all lowercase letters.

Part 5: On and Off Strings in Detail

With basic Replace characters, you can not selectively change material within a chapter. Instead, you have to break out your selections into separate chapters, Replace as required, then merge (or clipboard) the changed text back in to your original.

Contextual Replace's *on* and *off strings* save you a lot of extra work. In Part 2, we illustrated on and off strings by changing one word everywhere *except* in a heading, which began with $$c and finished with ($p). That's a good example of on and off strings that naturally occur in your text. The $$c characters are in your text to create a centered heading; the ($p) is there to make a new paragraph. By defining ($p) as the on string and $$c as the off string, you're getting double duty from those characters.

It's also possible to explicitly insert characters whose only purpose is switching Replace off and on. You want these characters to be very distinctive; three percent signs `%%%` makes a good on string, and three ampersands `&&&` makes a good off string. In part 7, we discuss the `FIX TEXT` contextual transformation chapter, which reformats textfiles into BEX chapters. When your textfile contains tabular material, FIX TEXT would cause havoc. As supplied on your BEXtras disk, FIX TEXT does not have on or off strings. When reformatting tabular textfiles, insert explicit on and off strings in the data and in FIX TEXT.

- ***Explicit, Non-printing On and Off Strings***

BEX's Grade 2 translator recognizes BEX's own underlining commands. So $$ub both begins underlining in print, and marks the beginning of braille italics. Likewise, $$uf finishes underlining and ends braille italics. When the translator encounters $$ub, it begins placing braille italics signs (dot 4-6, the period in screen braille) before every word. When the translator encounters $$uf, it stops placing italics signs. When the underlined passage is greater than three words, then the first word gets a double italics sign (two dot 4-6's or two periods in screen braille) and the last underlined word has a single italics sign.

Braille italics are used to represent various typeface changes in inkprint. But with some books, the braille reader must be able to distinguish between *three* typefaces: plain, italic, and bold. Placement of boldface indicators follows the

same rule as italics indicators. Three or less bold words each begin with the single boldface indicator `_.` underbar, period. The first word in a passage longer than three begins with the double boldface indicator `_..` underbar, two periods, and then the single boldface indicator precedes the last bold word.

Two of BEX's specific printer commands control boldface in print: $$eb begins boldface and $$ec returns to regular print. But the Grade 2 translator can't use $$eb and $$ec to place boldface signs. When the book you're transcribing contains extensive text that requires boldface indicators, it's time-consuming to place them all by hand. Combining the Grade 2 translator plus off and on strings, contextual Replace can automatically place boldface indicators.

It's a three step process. Enter $$eb and $$ec in your inkprint text. Then use basic Replace to transform $$eb to $$ub $$eb and $$ec to $$uf $$ec. The translator ignores the $$eb and $$ec commands; but because the $$ub and $$uf commands are there, it places italics signs as always.

The last step is using a contextual Replace transformation chapter with an on string of $$ub $$eb and an off string of $$uf $$ec. This transformation chapter changes `.` the italics indicator to a boldface indicator. The on and off strings prevent *every* italic sign from changing to boldface. This means that the only time italics signs are changed to boldface signs is when the italics signs appear between $$ub $$eb and $$uf $$ec.

BEX's formatter knows that braillers can neither underline nor do boldface. When you print your final grade 2 chapters to the braille previewer or an actual brailler, BEX suppress the action of all $$ub, $$uf, $$eb, and $$ec format commands.

Sample: Automatic braille boldface indicators

To get automatic braille boldface indicators, you must transform your data twice. In your inkprint data entry, type $$eb before boldface text and $$ec when you return to plain text. Before translation, use this basic Replace transformation:
Enter terminator: #
Find: <space> $$eb <space>#
Change to: <space>$$ub $$eb <space>#

Find: <space> $$ec <space># $$uf
Change to: <space> $$ec <space>#
Now you are ready to translate your inkprint into grade 2 or grade 1 braille. The translator places italics signs appropriately between all $$ub and $$uf underlining commands.

The following contextual transformation chapter changes the italics signs to boldface signs.
Enter terminator: #
Find: #
Contextual replace
On string: $$p-1 $$ub $$eb# $$uf
Off string: $$p-1 $$ec#
Find: <space>.#
Pattern: B^X#
Change to: _#
Find: #
Pattern: #
Continue? Y <CR>
The first find string character, <space>, when paired with a boundary B, defines the context at the beginning of a word. The period, when paired with boundary pattern code X, defines the first character of an italicized word in braille. There may be two italics signs or only one, but we actually don't care: we just want to insert the underbar that changes the italics sign to a boldface sign.

The pattern code meanings are for *print* data, which is important to keep in mind when writing contextual rules for braille data. For example, the P punctuation pattern code does not match braille punctuation, where a comma is represented with a screen braille digit *1* and the period is screen braille digit *4.*

Warning!

When you include the w wildcard pattern code in a transformation chapter that uses on and off strings, it's easy to "eat" your on and off strings by mistake. Suppose your off string is `#stop` and one rule contains three w departing pattern codes in a row. When Replace characters encounters `<space>#` it satisfies the rule, and the first two characters of your off string depart--and your replacement never turns off! The solution is to avoid the w wildcard. In this situation, it's safe to substitute the o

> departing pattern code paired with number sign. That way the off string is not affected by the replacement.

- ***Off Strings that Overlap Data***

While we don't recommend you find and change your exact on or off strings in the same replacement, you can modify text that's partly the same as your off strings. As BEX replaces characters, it works a character at a time. Suppose your off string is the first five letters of the alphabet: *ABCDE.* You *can* find and change `<space> ABC` because BEX hasn't encountered the full off string yet. But if your change deletes the *A, B,* or *C,* you can find yourself in deep trouble, because the full off string no longer exists. You have to be careful when pulling this trick; we only use this technique to insert material.

Sample: turning off underlining when $$h ends with ($l)

BEX's center-and-underline command, $$h can underline for quite a while if you forget to end the heading with a paragraph ($p) indicator. Using a transformation rule that partly overlaps the off string, you can insert the command to end underlining. Here's how:
Enter terminator: #
Find: #
Contextual replace
On string: $$p-1 $$h#
Off string: <space>$l <space>#
Find: d <space>$l#
Pattern: WbXX#
Change to: <space><space>#
Find: #
Pattern: #
Continue? Y <CR>
Starting to replace ...
The first character in the find string, when paired with the boundary pattern code W, stands in for last character of the heading. The next three characters are three-fourths of the off string, but the missing last fourth is enough of an overlap to make the transformation possible.

- ***Changing a Basic Transformation Chapter to Contextual***

The transformation chapter named `MAKE CON` on your BEXtras disk helps you transform a basic Replace transformation chapter into a contextual transformation chapter. It places its own on and off strings, which are the same character, then removes them at the end. MAKE CON clearly shows just how far you can push contextual Replace! We won't attempt to explain how it works, but it *does* work.

Here's how you use it. Suppose you have a basic Replace chapter named `HARPO` whose terminator is vertical bar. As supplied on the BEXtras disk, MAKE CON assumes that the basic Replace terminator is vertical bar. Edit HARPO and insert two characters at the beginning: `<control-A> <space>`

Quit HARPO, and run Replace characters. HARPO is your source chapter, the target chapter is named `ZEPPO` and the transformation chapter is MAKE CON. Once the transformation is finished, edit ZEPPO. Delete the `<control-A> <space>` at the start, and insert three terminators there. Advance to the end and add two terminators. Presto! ZEPPO is now a contextual transformation chapter.

When your basic Replace chapter uses a different terminator character, then modify MAKE CON. MAKE CON'S own terminator is slash. When your basic Replace terminator is any character *except* slash, simply use basic Replace to change vertical bar to your terminator.

Part 6: Using Contextual Replace

Congratulations! You've plowed your way through some pretty heavy material. In this Part of Section 6, we discuss various techniques to help you make the most of contextual Replace. First, we discuss BEX's error checking of pattern strings. Then we provide guidelines for preventing data salad. We present some thoughts on "elegance" in transformation chapters: how to get

the most done with the fewest rules. Finally, we offer some hints on creating contextual transformation chapters in the Editor, and making hard copy versions of contextual transformation chapters under development.

Here's a general hint to make contextual Replace more enjoyable: Obtain a memory expansion card, and configure an extended disk system with RAM drives. While RAM drives are always fun, they're especially useful when you're creating contextual transformation chapters. When BEX is reading and writing to a RAM drive, it only takes a minute or two to test the reliability of a transformation chapter. The less time required for you to test, the more likely that you actually perform the tests that guarantee smooth sailing.

Add comments to the end of your transformation chapters

When BEX encounters the three terminators that define the end of a contextual transformation chapter, BEX stops looking for more transformation rules. You can insert any text you wish after the three terminators. It's not always readily apparent what purpose a contextual transformation chapter serves, especially when it's been three months since you wrote it. We always add a sentence or two describing the function of the transformation chapter, as well as when it's best used, and the date it was last changed.

- ***Planning is All***

Even when you *think* you're a real contextual hotshot, it's possible to do truly idiotic things. While we were writing this Section, we thought we'd made a rule to automatically place BEX's underlining commands around single letters. It looked like this:
Enter terminator: #
Find: #
Contextual replace
On string: #
Off string: #
Find: <space>$p <space>#
Pattern: #
Change to: <control-P>#

Find: <space>$l <space>#
Pattern: #
Change to: <control-L>#
Find: <space> A <space>#
Pattern: Q^LQ#
Change to: $$p-1<space>#
Find: $$p-1*<space> A <space>*#
Pattern: XXXXBLQ^#
Change to: <space><space>#
Find: $$p-1*<space>*#
Pattern: #
Change to: $$p-1<space>#
Find: $$p-1<space>#
Pattern: #
Change to: $$p-1<space>#
Find: <control-L>#
Pattern: #
Change to: <space>$l <space>#
Find: <control-P>#
Pattern: #
Change to: <space>$p <space>#
Find: #
Pattern: #
The first two and last two rules work fine. The first two temporarily change the new-line and paragraph indicators to control characters, so the ($l) and ($p) are out of harm's way for the bulk of the rules. The last two rules undo the first two, transforming <control-P> and <control-L> back to ($p) and ($l). When you want to put particular strings of characters on a protected shelf during the Replace process, control characters are a great intermediate.

But the rules in between the first and last two did not work at all how we'd planned. Every time the letter *a* appeared as an article, as in "this move was a mistake," the article was underlined. Additionally, when a word ended with apostrophe *s* or apostrophe *t,* the last letter was split off and underlined. The third rule's pattern code of `Q^LQ#` was the source of the problem. We thought one rule could catch both single letters alone and single letters in quotes. But the final two characters in a word like *don't* also satisfied the pattern code. Fortunately, we used different source and target chapter names, so we just sighed and laughed at ourselves. (In fact, we were so

embarrassed that we just made the changes manually. Everyone has a bad day sometime.)

The moral is: Never underestimate the power of contextual Replace to change something you never considered. When you're *developing* a contextual transformation chapter, never use the same name for source and target. Only when you've *tested* your chapter with a broad variety of data is it safe to use the S naming method.

- ***Pattern String Errors***

When you start out writing contextual transformation chapters, it's easiest to type changes directly. BEX prompts you for the find, pattern, and change to strings, so you're less likely to get lost. BEX also provides some rudimentary error-checking for your pattern codes. This error-checking happens both when you type changes directly and when BEX is executing rules from a transformation chapter on disk. How you recover from the error depends on when it occurs.

When you directly type a character that's not one of the 34 valid pattern codes, then BEX beeps twice, prompts `Illegal character in pattern string` and throws away both the find and bogus pattern strings you've just entered. BEX reprompts `Find:` to give you a chance to do it right.

When you directly type the wrong number of pattern codes in your pattern string, then BEX beeps, prompts `Character counts are not equal` and again discards the most recent find and pattern strings. Then BEX reprompts `Find:` to give you another chance. Finally, when your pattern string contains all boundary codes, but it doesn't contain the caret, then BEX beeps twice, prompts `Pattern string needs caret or one departing code` and ignores the most recent find and pattern strings. Again, BEX reprompts `Find:` and waits for your next attempt.

Enter intentional error to get a second chance

You can take advantage of these error-checking routines when you realize that your find or pattern strings are simply wrong. Suppose you enter the

wrong characters in the find string; type `<space><terminator>` as a pattern string and BEX rebels, giving you an opportunity to try again.

When errors occur with a transformation chapter from disk

Whenever BEX finds problems with a pattern string, it always issues these error messages--even when the problems occur with a transformation chapter loaded from disk. When you first begin to write contextual transformation chapters in the Editor, you're guaranteed to run into this situation at least once. When your pattern strings are too long, too short, or contain invalid characters, then BEX issues one of the error messages, *right in the middle of the act of replacing.* Any and all correct transformation rules are executed for each BEX page until BEX encounters the faulty pattern string. Then BEX saves the first page, and goes on to the next page, again reporting the error when it encounters the faulty pattern string. In short, BEX executes as many rules as it can, saves the target chapter, and reports the number of times replaced.

An example may make this clearer. Suppose you created the following `DUMB TRANS` transformation chapter:

```
#### #b#@#99#===# ###
```

The first rule changes every <space> to an at-sign. The second rule's pattern string does not contain any of the 34 valid pattern codes. You ask BEX to use this DUMB TRANS chapter to Replace characters in the RESUME chapter. Once you specify the different target chapter, like RESUME-TEST, and commence replacing, you hear an enormous number of clicks as BEX changes all the spaces to at-signs. Then you hear two beeps and the `Illegal character in pattern string` error message. Next, you hear and see a BEX program function that's usually hidden from you; BEX saving the binary file of the target chapter page. The last thing you hear is `Replaced 408 times` after BEX has saved the imperfectly replaced RESUME-TEST chapter. If the RESUME chapter had two pages, you would hear the error message twice.

BEX issues this error message at the exact point when it tries to execute the transformation rule with the faulty pattern string. It's instructive to take a look at these half-baked target chapters; you may be able to see exactly where you erred. Once you're comfortable with editing transformation chapters,

you can edit a fault chapter and insert two extra terminators after a rule to end the transformation chapter prematurely. Run Replace again; if you don't get the error messages, then delete the extra terminators and insert them in the next rule.

The fastest way to type changes directly

Whenever you specify the Ready chapter by its right bracket name, BEX assumes that the Ready chapter contains data, even if it's empty. You can take advantage of this fact to quickly create a contextual transformation chapter. Specify the Ready chapter as both source and target chapter and type your changes directly. After you press <CR> at the `Continue? Y` prompt, BEX almost instantly tells you `Replaced 0 times` and allows you to save the transformation chapter. You can save it to disk, or even in the Ready chapter, and then edit the transformation chapter you've just created.

● *Elegance and efficiency in transformation chapters*

The time Replace characters requires to process your chapters depends on the number and nature of rules in your transformation chapter. You want to have as few rules as possible, but you also want to transform as many characters as possible with each rule.

When we introduced the L and P pattern codes in Part 3, we described transformation rules that changed single letters in quotes to underlined single letters. The transformation rules there more-or-less got the job done, but there's a more elegant solution. The following set of rules is shorter and handles *every* combination of punctuation touching the quoted letter, even when the source text contains `("a," "b," and "c").`

Enter terminator: #
Contextual Replace
On string: #
Off string: #
Find: "a"<space>#
Pattern: xLXB#
Change to: <control-T><space>#
Find: "a".#
Pattern: xLXP#

Change to: <control-T><space>#
Find: "a."<space>#
Pattern: xLPXB#
Change to: <control-T><space>#
Find: <space><control-T>#
Pattern: xx#
Change to: <space>#
Find: $$ub<space>a"<space>#
Pattern: XXXXBLxB#
Change to: <space>#
Find: $$ub<space>a".#
Pattern: XXXXBLxP#
Change to: <control-T><control-T>#
Find: $$ub<space>a."<space>#
Pattern: XXXXBLPxB#
Change to: <space>#
Find: #
Pattern: #

The first three rules change *any* initial quote touching a single letter to <control-T>$$ub<space> even though many situations won't require the initial touching token. The point here is that when you *don't* need the touching token, it's obvious. When underlining does not begin mid-word, the data looks like `<space><control-T>` and the fourth rule deletes this unnecesary <control-T>. The moral is: when a few rules can insert *more* than you need, you can then delete the "extra" later on in the transformation chapter.

The fifth, sixth, and seventh rules demonstrate find and pattern strings that search for the *minimum* required. It doesn't really matter whether the underlining begins mid-word or not, so the find string for these three rules begins with the initial dollar sign of $$ub.

Also in Part 3, we demonstrated the o pattern code with two rules that delete all JustText typesetting commands. These commands always begin with left brace and end with right brace. The two rules delete every character between a left brace and a right brace, then delete the left brace-right brace pair that's left. While this approach is remarkably economical, it's also quite *slow.* Replace characters ends up comparing every character in the source chapter

against the transformation rule at least four times, so it requires quite a while to strip out the commands.

A much more efficient procedure would use a few more rules:
Find: {}}}}}#
Pattern: Xooooo#
Change to: #
Find: {}}}#
Pattern: Xooo#
Change to: #
Find: {}#
Pattern: Xo#
Change to: #
Find: {}#
Pattern: xx#
Change to: #
The first rule deletes a six-character JustText command in one swipe; the second deletes a four-character command; the third deletes any commands that are left, and the fourth rule again deletes the residual braces. The second version accomplishes the same task in around half the time as the first.

• *Creating Contextual Transformation Chapters in the Editor*

Whenever you type changes directly, make a habit of saving the transformation chapter, then editing it. (It's a good time to add the comments we mentioned above.) The more exposure you have to these chapters, the easier it becomes to create them from scratch. As we mentioned in Part 2, contextual transformation chapters contain every keystroke entered between the `Enter terminator:` and `Continue? Y` prompts. Whenever possible, use <CR> as your terminator. It makes moving around in the Editor much simpler since you can control-G and control-R between strings. To move to the next rule, you enter control-A 3 control-L.

The one essential prerequisite is knowing how many terminators you need. Every transformation rule has three terminators. The total number of terminators in a contextual transformation chapter is three times the number

of rules, plus six. Four of that extra six are at the start of the chapter, and the last two finish up the list of rules.

Here's a quick review of the structure of a contextual transformation chapter. All contextual transformation chapters begin with two terminators. The on string, if you have one, goes between the second and third terminators. The off string, if you have one, goes between the third and fourth terminator. After these four terminators, the list of transformation rules starts.

Every rule follows the pattern of find string, terminator, pattern string, terminator, change to string, terminator. Remember the pattern string shortcut: an empty pattern string specifies an exact departing match for all characters in the find string.

The list of rules always ends with at least three terminators in a row. When your last change to string is empty, then the chapter ends with four terminators in a row. When your last pattern string uses the single terminator shortcut, *and* your last change to string is empty, then your chapter ends with five terminators in a row.

Sample: counting terminators in transformation chapters

Basic Replace characters can help you be sure that you have the correct number of terminators. In this sample, we are creating a `WHITE` contextual transformation chapter; its terminator is the vertical bar:
Main: R
Replace
Chapter: WHITE <CR>
Chapter: <CR>
Target chapter: WHITE <CR>
Use transformation chapter: <CR>
Enter terminator: <CR>
Find: |<CR>
Change to: |<CR>
Find: <CR>
Continue? Y <CR>
Starting to replace ...
All this replacement does is change the vertical bar terminator to itself. When it's finished, BEX announces `Replaced # times` and lets you save the

transformation chapter. That # corresponds to the number of terminators in the transformation chapter; when it is divisible by three, then you have probably put terminators in all the right places.

Use <CR> as terminator while developing chapters

Even if you need to use the <CR> character as data in your rules, you can write a "development" version of the transformation chapter that uses <CR> as terminator. In addition to making it easier to move around in the Editor, when your terminator is <CR> it's much easier to make a hard copy printout of your work-in-progress.

Here's how: in your development version, use a different unique character to stand for <CR>. Suppose your transformation chapter does not contain any tildes. You put tilde in the development version wherever you want <CR> in your final transformation chapter. Once all your development work is complete, you use basic Replace characters on the development chapter to create the final version. In this case, you change the <CR> terminator to another character, like vertical bar, then change tilde to <CR>. Suppose your development chapter is named `DEV` and `WHITE` will be the name for the final transformation chapter. Once the DEV chapter is ready, you go through this dialogue:
Main: R
Replace
Chapter: DEV <CR>
Chapter: <CR>
Target chapter: WHITE <CR>
Use transformation chapter: <CR>
Enter terminator: #
Find: <CR>#
Change to: |#
Find: ~#
Change to: <CR>#
Find: #
Continue? Y <CR>
Starting to replace ...

Don't get dizzy! Using Replace characters to help you create contextual transformation chapters is so self-referential that you can feel the ground tilting beneath your feet.

Finally, the Clipboard can be very helpful while creating contextual transformation chapters in the Editor. You can type the find string, then copy and insert it as the basis of the pattern string. Overwrite each of the find string letters with the pattern code you've chosen. When you write a series of rules that address slightly different boundary contexts, copy one rule then use it as the template for the next rule.

- ***Making Hard copy Printouts***

Because the pattern codes are so cryptic, it's often difficult to hold all the interactions in your head. In our experience, it's a *lot* easier to do your design work in a hard copy medium. As we mentioned in the STRIP DOLLAR sample in Part 3, the order of the transformation rules is often crucial to their success. When you're working in hard copy, you can think up all the possible rules, then shuffle them around to get the optimal order.

To make useful hard copy print or braille editions of your transformation chapters, you must change your terminator into a two-character sequence; a unique character plus <CR>. The <CR> makes each string a new line. The unique character $$vt $$vp aM6:42 alerts you to an empty string, which

would otherwise be a blank line. When <CR> is used for data, then you must replace it with a different, printable character. For example, when your transformation chapter contains neither the vertical bar nor the lowercase letter *z,* change every appearance of <CR> as data to lowercase *z.* Then change your terminator character to vertical bar followed by <CR>.

You must also change any other control characters in your rules to printable characters, or you may find your printer doing some pretty strange things. When the spaces in your rules are important, change the space character to a printable character as well.

When your transformation chapters contain $$ commands, you have to "defuse" them in order to make a useful hard copy. Inserting $$z at the beginning of the transformation chapter does the trick. Your printer or brailler still receives the hard <CR> at the end of each string, so each string appears on a separate line.

Sample: printable version of FIX TEXT

Here's how you make a printable version of the `FIX TEXT` contextual transformation chapter on your BEXtras disk. It's a pretty stinky sample, since FIX TEXT contains <CR> s as data, several other control characters, and BEX $$ commands. (We have an ulterior motive for picking this sample; in Part 7 of this Section, we analyze how FIX TEXT works.)

First, edit the chapter and see what's there. To make a printable version, you must replace non-printing characters with printing characters. But you must choose printing characters that are not already present in the transformation chapter. If you don't then you would not be able to make sense of the print out.

The terminator is slash. There are four control characters: <CR>, <control-J>, <control-H>, and <control-T>. The simplest system is to substitute the plain letter for the control letter. Locate for lowercase *m, j, h,* and *t:* you get four beeps because those letters aren't in FIX TEXT. Great! That means you can substitute *m* for <CR>, *j* for <control-J>, *h* for <control-H>, and *t* for <control-T>.

There are many meaningful spaces in FIX TEXT. A good way to represent spaces is with the underbar. First, check to see if it's already used. When you locate for the underbar you get several hits, so that won't work. Try the asterisk character--none are present in FIX TEXT, so you can use the asterisk to represent spaces. You're ready to type these changes:
Main: R
Replace
Chapter: FIX TEXT <CR>
Chapter: <CR>
Target chapter: PRINTABLE FT <CR>
Use transformation chapter: <CR>
Enter terminator: <control-H>
Warning! Arrow keys used as data!
Enter terminator: #
Find: <CR>#
Change to: m#
Find: <control-H>#
Change to: h#
Find: <control-T>#
Change to: t#
Find: <control-J>#

Change to: j#
Find: /#
Change to: /<CR>#
Find: <space>#
Change to: *#
Find: #
Continue? Y <CR>
Starting to replace
Replaced 155 times
Save transformation chapter: SHOW CODES <CR>
Remembering back to User Level Section 8, the left and right arrow keys are generally used to correct your strings as you type them directly. When you need to include <control-H> and <control-U> in your strings, you press the left arrow at the `Enter terminator:` prompt, as shown above. From that point, you have to type your rules perfectly, since you can't use the left and right arrow to fix mistakes.

Before you can actually print the `PRINTABLE FT` chapter, you must edit it and insert the $$z command at the very beginning.

Part 7: Inside FIX TEXT and SP2

`FIX TEXT` and `SP2` are two contextual transformation chapters on your BEXtras disk. In this Part, we examine what they do and how they do it.

SP2 deletes all occurrences of two spaces, except when they appear at the end of a sentence. FIX TEXT reformats data that's been created by Input through slot and Read textfile. This data generally is formatted as if it's been printed to disk; more details on Input through slot are available in User Level Section 12. Textfiles are discussed in Learner Level Section 12 and User Level Section 10.

- ***Inside FIX TEXT***

Copy the FIX TEXT transformation chapter before you edit it, so you don't have to worry if you change something by mistake. When you have a hard copy print or braille device, print out the PRINTABLE FT chapter you created by following the instructions in Part 6.

What FIX TEXT does

FIX TEXT changes a blank line plus many spaces to lines beginning with the BEX format command $$c; changes two <CR> s to a BEX paragraph indicator; changes a single <CR> to a space; changes two spaces to one; and attempts to place BEX's underline begin and finish commands where ne $$.

Most of its work could be accomplished with plain Replace characters. However, since the rules for placing underlining commands had to be Contextual, all of FIX TEXT had to be written in contextual form. As supplied on the BEXtras disk, FIX TEXT does not contain on or off strings. After we examine how the rules work in detail, we discuss when on and off strings would be appropriate.

Rule-by-rule through FIX TEXT

The 16 rules in FIX TEXT are divided into two broad categories. The first six rules deal with <CR> s and spaces. In these rules, each find string is followed by two slashes; the pattern string shortcut that means an exact, departing match for every find string character.

The first rule deletes linefeeds, also known as <control-J> s. Many IBM programs end each line with two characters, <CR><control-J>, while most Apple programs end each line with just <CR>. By deleting the <control-J> s first, subsequent rules apply equally to both type of line endings.

The second rule changes two <CR> s followed by 11 spaces into two <CR> s followed by BEX's $$c centering command. We settled on 11 spaces as a good solution through trial and error. If the number of spaces is too small, then the rule would be satisfied by the beginning of every paragraph. When your source material contains deeply indented paragraphs then FIX TEXT would think every line should be centered. But if the number is too large, then only very short headings satisfy the rule, and you would need to place the $$c commands manually.

Rule three changes three <CR> s to two <CR> s, and the rule four turns two <CR> s into BEX's paragraph ($p) indicator. We assume here that the source text always has at least one blank line between paragraphs. Now that we've defined the paragraphs, all other <CR> s are superfluous; rule five changes any <CR> s that are left into a space.

The sixth rule changes two spaces to one space. When you want to have two spaces at the end of sentences, use the SP2 transformation chapter on the BEXtras disk after you've used FIX TEXT.

The remaining ten rules cope with underlining and are fully contextual. To make sense of these rules, you must first analyze what underlined text looks like. We assume that the source computer accomplishes underlining in the same way BEX does: print a character, then a backspace <control-H> command, then the underbar character. The combination of <control-H> and underbar creates the underlining.

There are three possible contexts: when underlining begins; when underlining ends, and underlining "midstream."

Understanding midstream underlining provides the key: <control-H>, underbar, character, <control-H>, underbar, character, over and over. We want to delete the <control-H> s and underlines in this context.

The pattern for the beginning of underlining is defined as *not* like midstream underlining. When underlining starts, the first four characters are a character that's not an underbar; then the underlined character; <control-H>; underbar. We want to place BEX's $$ub UNDERLINE begin command here.

When underlining ends, the final characters are: last underlined character; <control-H>; underbar; then a word delimiter, like <space>, <CR>, or punctuation. We want to place BEX's $$uf underline finish command here. If the word delimiter is not a <space>, then we want to place <control-T> s around the underline finish command. (The <control-T> *touching token* is explained in Section 5, Part 1. It can replace the initial and final spaces in BEX $$ format commands.)

Rule seven searches for the beginning of underlining: it inserts dollar, dollar, lowercase *u*, lowercase *b*, <space> when the data does not follow the midstream pattern. The combination of the find and pattern strings specifies

any character *except* underbar, followed by three boundary characters: the total wildcard W, then exactly <control-H> and underbar. The caret appears after the boundary O, so the-mand is inserted before the first underlined character.

The seventh rule inserts the-mand right next to punctuation. Suppose you underline the first syllable of the word *despair* enclosed in parentheses; your source data looks like this:
(d <control-H>_e <control-H>_s <control-H>_pair)
the first four characters match the seventh rule, so your data is transformed to
(<space> d <control-H>_e <control-H>_s <control-H>_pair)
The eighth rule changes the `$$ub<space>` to `<control-T><control-T>` The initial <control-T> means BEX's formatter recognizes the underline begin command; the final <control-T> means the Grade 2 translator recognizes the end of the $$ command, so the following characters are translated.

Rule nine deletes all <control-H>-underbar pairs that occur *midstream.* The initial <control-H> and underbar are paired with lowercase x, so they depart from the target chapter. Because the boundary patterns are total wildcard followed by exactly <control-H> underbar, this rule does not match the last underlined character in a word.

The tenth rule matches the last underlined character in a word, because the boundary character in the pattern string is D, standing for space or <CR>. The <control-H> and underbar are replaced with space, dollar, dollar, lowercase *u,* lowercase *b.* BEX's underline finish command requires a final space or <CR>; it's supplied by the character in the source chapter that matched boundary code D.

At this point, the only situation where <control-H>-underbar pairs remain in the source chapter is the pattern of last underlined character, <control-H>, underbar, punctuation. Rule 11 matches this situation. The question mark in the find string is paired with the E boundary code, meaning match everything *except* a <space> or <CR>. The <control-H> and underbar that depart are replaced with the underline finish command, preceded and followed by the <control-T> touching token.

Rule 12 gets rid of underline begin immediately followed by underline finish. This pattern appears when the formatter that did the underlining in the source chapter suppressed underlining for the space between words.

The remaining four rules undo the effect of rule 11 for four punctuation marks. We don't need to bother with the touching token for period, comma, semicolon, or colon because BEX's selective punctuation format command $$sp prevents underlining these four punctuations marks when they appear at the end of a word.

Turn Off FIX TEXT for Tables

When your source text contains tables or columns, FIX TEXT'S rules would create a disaster. Suppose you have a table with 6 columns and 15 rows printed to an 80-character carriage width. If it's single-spaced, FIX TEXT changes the <CR> s that define the end of each line to a space--and presto, you lose the line-oriented structure of the data. FIX TEXT would also change the multiple spaces between each column into a single space. Since FIX TEXT is a contextual transformation chapter, you can simply switch off the execution of its rules for the line-oriented portions of the text.

Since FIX TEXT as supplied on the BEXtras disk does not contain on or off strings, you can insert whatever characters you prefer. The terminator in FIX TEXT is the slash. The first slash in the chapter announces "Slash is the terminator." The second slash announces, "This is a contextual rather than plain transformation chapter." The third and fourth slashes are place holders for on and off strings respectively.

For the sake of example, choose `@@text` for the on string and `@@lines` for the off string. Edit a copy of FIX TEXT. Place your cursor on the third slash; press control-I, type the six characters `@@text` and press right arrow. Now press control-I, type the seven characters `@@lines` and press right arrow again.

Place these on and off strings where appropriate in your source chapter before using FIX TEXT. Whenever a transformation chapter contains on or off strings, Replace begins *off.* None of the rules are executed until the program encounters the first on string. So you must insert `@@text` at the first occurrence of non-tabular material in your source chapter. When you encounter a table, put `@@lines` immediately before it and `@@text`

immediately after it. Now you can use Replace characters on the modified source chapter, with the modified FIX TEXT as your transformation chapter; it won't destroy the format of tables.

To really automate the process, you write a transformation chapter specifically designed to reformat tables; in this chapter, make `@@lines` the on string and `@@text` the off string. The last step is removing `@@lines` and `@@text` from the transformed chapter. Naturally, you write a transformation chapter that just deletes the on and off strings.

- ***When to Use SP2 and What It Does***

Correctly formatted braille and typeset print always contains just one space at the end of a sentence. On the other hand, typewritten print contains two spaces at the end of a sentence. When you enter text with two spaces, it's child's play to delete the extra space before you make braille or typeset text. But when your original text contains just one space after a sentence, it's massively boring to insert the extra space by hand to make correctly formatted print.

The contextual transformation chapter named `SP2` is on the BEXtras disk. (We originally thought of calling it SPACE THE FINAL FRONTIER, but that's a lot to type.) Use SP2 whenever you want to create appropriately formatted typewriter-like print output. The rules in this chapter delete all occurrences of two spaces, then add two spaces after sentences, then delete two spaces after abbreviations.

Before you examine SP2 in the Editor, make a copy. Its terminator is <CR>. To make a print or braille hard copy, follow the suggestions in Part 6; replace space with some printable character. The discussion that follows assumes you have hard copy in hand.

Before we analyze each of the 24 rules in this chapter, let's sit back and think about how sentences appear in text. Since we're going to be talking about sentences a lot, we use a shorthand: *S1* means the first sentence, and *S2* means the second. SP2 only deals with the spaces that come between one sentence and the next sentence, or in shorthand, the spaces between S1 and S2. When S1 and S2 are separated by a <CR>, ($l), or ($p), then the number

of spaces between them is irrelevant. Any number of spaces at the end of S1 just become trailing spaces at the end of a line. What other contexts can exist for this transition?

The first character in S2 is either an uppercase letter or punctuation, like a single or double quote. It's unlikely that S2 begins with a digit, as most style books frown on this. (When a number comes first in a sentence, most style books recommend you spell out the number in words.) The last character in S1 is always punctuation.

However, we can't define that punctuation with the P pattern code, because that code includes a number of symbols that can appear *within* a sentence. The percent sign and the asterisk are just two samples of symbols frequently found in the middle of a sentence. Without the P code, we must develop a list of possible sentence-ending punctuation--and that's why SP2 contains so many rules.

The final analytical challenge concerns abbreviations. The period character does double duty. Most sentences *end* with period, but period often appears *inside* a sentence in abbreviations like *Mrs.* and *Still.* SP2 tries to define several patterns of letters that abbreviations can follow; SP2 attempts to delete two spaces when they follow these patterns and end with a period.

The next-to-last sentence in the previous paragraph is a sample of a situation where SP2 fails. The period serves as the end of the *Still.* abbreviation *and* as the end of the sentence. There are several other contexts where SP2 is overwhelmed: a computer program is no match for a human being when it comes to subtle tasks like deciding where a sentence ends. We point out these problems as we examine SP2 in detail.

Rule-by-rule through SP2

The terminator in SP2 is <CR>, making it easy to move through the rules in the Editor. Although SP2 is contextual, it does not use on or off strings. The first rule changes two spaces to one: all subsequent rules insert an extra space where appropriate. The rules 2 through 15 parallel the rules 16 through 26. The first bunch define S2 as beginning with an uppercase letter; the second bunch define S2 as beginning with punctuation followed by uppercase letter.

Rule two defines the most common arrangement of characters between S1 and S2: period, space, uppercase letter. The single departing pattern code b is changed to two spaces. The character preceding the period could also be punctuation, for example, the British method of placing the period outside quoted material. Rule 16 is parallel to rule 2--S1 is the same and S2 changes. For rule 16 S2 begins with punctuation followed by uppercase letter.

Rule three defines S1 as ending with question mark instead of period; rule 17 shows S1 ending with period and S2 beginning with punctuation. Rules four and 18 show S1 ending with exclamation mark.

Rules five through 12 and 19 through 26 specify various sentences that end with two punctuation characters.

Rules 13 through 15 are the abbreviation-handlers. Thirteen deletes the extra space after the honorary *Mrs.* regardless of the case of the letters *r* and *s.* Rules 14 is more general: the pattern codes match any two-letter abbreviation where the first letter is uppercase. So in addition to Dr. these rules match *MR., St., AV., Ln.* and literally hundreds of others. Because the first pattern code is boundary Q, this rule matches an abbreviation with touching punctuation. Rule 15 matches a single uppercase letter, like those found in lettered outlines.

There are *many* possible abbreviations, and these three rules don't come anywhere near matching all of them. But, it's important to compare the benefits gained with the effort required to write hundreds of rules. When SP2 fails, all that happens is there's an extra space in your text, which is not life-threatening.

The final rule matches S1's that end with BEX's underline finish command. The space before the first dollar sign is printed, but the space after the lowercase *f* is not. The final rule adds that extra space.

Part 8: Contextual Replace Reference

Enter terminator at first `Find:` prompt to get into contextual Replace. Number of terminators in contextual Replace transformation chapter must be

divisible by three. Total number of terminators is three times the number of transformation rules plus six. First two characters are terminators, then (optionally) on string, terminator, off string, terminator. When on string is present, then contextual Replace begins *off.* End list of rules with three terminators.

Lowercase Departing Pattern String Codes

- a - Alphanumeric; either letter or numeral
- b - Blank; space
- c - Control character; includes <CR>
- d - Delimiter; space or <CR>
- e - Everything which is *not* a delimiter
- i - exact match with find string partner except for Ignoring capitalization
- l - Letter, regardless of case
- n - Numeral
- o - matches anything Other than find string partner
- p - Punctuation and symbols
- q - punctuation, symbols, space, or <CR>
- s - Small letter, always lowercase
- u - Uppercase letter
- w - total Wild card (matches everything)
- x - eXact match with find string partner

Uppercase Boundary Pattern Codes

- A - Alphanumeric; either letter or numeral
- B - Blank; space
- C - Control character; includes <CR>
- D - Delimiter; space or <CR>
- E - Everything which is *not* a delimiter
- I - exact match with find string partner except for Ignoring capitalization
- L - Letter, regardless of case
- N - Numeral
- O - matches anything Other than find string partner
- P - Punctuation and symbols
- Q - punctuation, symbols, space, or <CR>
- S - Small letter, always lowercase
- U - Uppercase letter
- W - total Wild card (matches everything)

- X - eXact match with find string partner

Special Pattern Codes

Pattern string shortcut - Entering terminator alone at `Pattern string:` prompt means an exact match, equivalent to pattern code of all lowercase x

- ^ - (caret) insert change to string here, only use when no departing codes present
- Y - behaves like boundary L but forces uppercase
- Z - behaves like boundary L but forces lowercase $$ml0 $$i2
- Pattern code relationships
- D equals B plus <CR>
- Q equals D plus P
- L equals S plus U
- A equals L plus N
- E equals W minus D
- W equals all other codes

Punctuation as defined by pattern code P

When the Echo uses a different term from standard practice, that term is included in parentheses):

- ! - exclamation
- " - double quote
- # - number sign
- $ - dollar sign
- % - percent
- & - ampersand (or and)
- ' - apostrophe
- (- left parenthesis
-) - right parenthesis
- * - asterisk (or star)
- + - plus
- , - comma
- − - hyphen (or dash)
- . - period
- / - slash
- : - colon

- ; - semicolon
- < - less-than
- = - equals
- > - greater-than
- ? - question mark
- @ - at-sign
- [- left bracket
- \ - backslash
-] - right bracket (or ready)
- ^ - caret (or up-arrow)
- _ - underline
- ` - grave accent
- { - left brace
- | - vertical line (or vertical bar)
- } - right brace
- ~ - tilde

7 Auto Chapters

You can tell BEX to save a session at the keyboard as an *automatic procedure chapter* for future use. To use an auto chapter, you press control-A at the Main, Second, or Page Menus. BEX asks for the auto chapter's name, and then interprets each character in the chapter as if it were typed on the keyboard.

Part 1: Overview

Auto chapters are like a piano roll for a player piano. You can use auto chapters to perform standardized, multi-step operations. You can use an auto chapter to perform an operation you do frequently. Even at the Learner Level, you can enter control-A to use an Auto chapter. In Learner Level Section 10, we described the `VOICE ON` and `VOICE OFF` auto chapters. But you can only *create* auto chapters at the Master Level.

Auto chapters are particularly useful when you configure an extended disk system. When BEX is running from a RAM drive or the Sider, you can direct repeated processing for several disks of data.

As an example, we made heavy use of auto chapters when writing this manual. The inkprint text of this manual is stored on the Sider hard disk. Creating the braille version requires four steps: transforming generic format information to braille-specific format information with Replace characters; then translating the resulting target chapters, then replacing again to fix minor translator problems, and finally proofreading the grade 2 text. We get everything ready at night, tell BEX to use an auto chapter that performs the first three steps, and our data is waiting to proofread in the morning.

Part 2: Creating an Auto Chapter

You can only create or invoke an auto chapter at one of the three menu prompts on BEX's Main side. You can't invoke an auto chapter at the `Naming method:` prompt or any other BEX prompt. You can't create or use an auto chapter at the Starting Menu.

An auto chapter cannot in turn call another auto chapter; you can't nest your operations. Finally, an auto chapter cannot include use of option A - Auto print from VersaBraille on the Main Menu.

While creating an auto chapter, you can use the Editor, move to any Main side menu and use any option there, and leave BEX with a Q to use DOS or Applesoft commands.

- ***Maximize flexibility***

To make an auto chapter reusable, avoid using specific chapter names at source and target chapter prompts. When you scan a drive for chapters, BEX always asks `Use entire list? N` even if there's only one chapter on the disk. In Section 4, we explain the period code for the target chapter naming method. When you place a period between the drive number and the

naming method letter(s), BEX can correctly interpret a naming method even if it's prompting for `Target chapter name:` instead. These two features make it easier to write auto chapters that work the same way whether there are one or 10 source chapters.

• *Commands you use to create an auto chapter*

- 1. Press control-R at a Main side menu prompt: you are now working in *remember mode.* BEX responds `Start remember mode` and begins storing every key you press in a special buffer.
- 2. Do exactly what you want BEX to remember; BEX responds normally to all your commands. For every key you press, there's a small click to remind you that BEX is remembering. BEX can remember up to a maximum of 2048 characters.
- 3. You must finish the procedure at one of the three Main side menu prompts. Once you're done, press control-S and BEX responds with `Auto chapter:` BEX copies the buffer full of keystrokes to the chapter name you provide.

SlotBuster:	Control-R is the SlotBuster's *enter line review* command. You can't type control-R to start remember mode, because SCAT gets the command first. Pressing asterisk (shift eight) is the same as pressing control-R.
64K Apple:	The limit on the keystrokes in an auto chapter is 256.

After you press control-R, BEX remembers almost every keystroke. When you type a chapter name, then use the left and arrow to correct it, all the letters you type plus the ;H> and ;U> commands are remembered in your auto chapter. BEX does not remember control-X to shut up the Echo. (That's because BEX never gets the control-X; TEXTALKER grabs it first.) BEX does not remember when you press to cancel an option in progress. We discuss this further under Shortcuts, below.

• *Sample: Creating and using an auto chapter*

You want to automate killing the contents of the Ready chapter. As mentioned in Section 2, when you kill the Ready chapter, you are sure to delete any extra page files on disk. The first time through, you do this:

```
Main: control-R
Start remember mode
Main: S
Second: K
Kill chapters
Chapter: ] <CR>
Chapter: <CR>
O K to proceed? N Y <CR>
Chapter ] done
Second: J
Main: control-S $1 Auto chapter: 1ZAP <CR>
```

Now the next time you want to kill the Ready chapter, you do this:

```
Main: control-A
Auto chapter: 1ZAP <CR>
```

and sit back and watch the fireworks. Notice that the first keystroke in the ZAP auto chapter is *S*. When you're already at the Second Menu, this has no effect. But because that *S* is there, you can invoke this auto chapter at the Main and Page Menus as well.

- ***Canceling remember mode***

When you regret something you've done after you press control-R, you can cancel remember mode by pressing control-R a second time. BEX responds `Cancel remember mode` and abandons the keystrokes it was remembering.

In addition to canceling remember mode when you press control-R a second time, BEX automatically cancels remember mode whenever it encounters a disk error. Suppose you don't have a disk in your default data drive, and you do this:

```
Main: control-R
Start remember mode
```

```
Main: D
Which drive? 2 <CR>
Cannot read the disk
Disk error!
Cancelling remember mode
Main:
```

This is a merciful feature: you don't want BEX to run away wildly with your data if it can't write to disk!

Shortcuts for Creating Auto Chapters

You may find creating and using auto chapters trickier than you expect. Since auto chapters invite you to perform lengthy unattended operations, you must find a way to avoid changing disks midstream. This goes hand in hand with the flexible source and target naming methods, and especially the ability to have more than two disk drives. When you run BEX from a floppy drive, your auto chapter operations can only change chapters on drive 2. But when you run BEX from a RAM drive or the Sider, the sky's the limit.

As discussed in Learner Level Section 13, you can press to cancel grade 2 translation, Replace characters, or Print. When you are in remember mode, BEX does not remember the . When you're remembering one of these activities, you can specify source and target chapters, then press as soon as BEX starts translating, replacing, or printing.

You can create an auto chapter for a time-consuming operation by performing the operation on a disk containing a short dummy chapter. Use the period code when naming the target chapter. The auto chapter you save contains the same keystrokes you need to do a disk-full of chapters, provided you set things up to avoid disk full errors.

We recommend that your auto chapter begin with the command to get to the right menu. That way you can invoke the auto chapter from any Main side menu.

Part 3: Using an Auto Chapter

While you can only create an auto chapter at the Master Level, you can use one at any level. You could create an auto chapter that demonstrates using BEX for a beginner. At all Levels, you invoke an auto chapter by pressing control-A at any menu on the Main side. How BEX responds depends on your level--at the Master Level, it's simply `Auto chapter:` At all levels, you must type in the name. You *can't* scan the disk for an auto chapter. Once you have given the chapter name, your typing at the keyboard is ignored, because the Apple is getting its input from the auto chapter. The Apple speaker clicks each time the Apple accepts a character. You also hear all of the messages and prompts that you would hear if you were typing things in.

There are two ways to suppress Echo speech during the execution of an auto chapter. The simplest is to press control-X as soon as you hear the first prompt. Although BEX is only paying attention to the keystrokes in the auto chapter, TEXTALKER still responds to control-X. We explain another method to suppress speech when we discuss editing auto chapters in Part 4.

When you want to stop an auto chapter in process, you have two choices. Pressing Control-Reset stops an auto chapter dead in its tracks. You should never press Control-Reset when BEX is writing to disk, however.

Because auto chapters always stop when BEX gets a disk access error, another way to stop an auto chapter is to pull the disk out of a drive. The next time BEX needs to read from disk, DOS returns an I/O ERROR that stops the execution of the auto chapter. Never do this when BEX is writing to disk. Whether you use Control-Reset or create a disk error, the auto chapter is stopped, not paused. Typing `RUN` does not restart it.

Part 4: Editing and Modifying an Auto Chapter

The final character in an auto chapter signals BEX to start paying attention to the keyboard again. When you edit an auto chapter, this final character *seems* to be a <Del> character, but it actually isn't. It's a *high bit set* character that you cannot type in the Editor. We use the highly scientific term *magic <Del>* to refer to the last character in an auto chapter.

When you edit an auto chapter, you must be very careful not to delete this last character. If you do delete it by mistake, then edit the VOICE ON chapter on the main side of BEX and copy its magic <Del> character to the clipboard. Then insert the magic <Del> character at the end of the auto chapter you're creating.

Once you're very familiar with BEX it's possible to create auto chapters from scratch in the Editor. However, it's very easy to forget just one <CR> that throws everything off. It's quicker to modify existing auto chapters.

Every key you press at a BEX menu is interpreted as uppercase. This is also true when BEX is getting its keystrokes from an auto chapter, so don't worry about case when you type auto chapter information in the Editor.

- ***Auto chapters and the Echo***

In Section 8 we explore BEX's control-B commands that let you control all four BEX channels at any point. One pair of commands controls the voice channel: control-B V A activates the voice channel and control-B V D deactivates it. When you want to turn off Echo speech during the execution of an auto chapter, you place these commands in it. When you depend on Echo speech, you don't want to issue these commands as you're creating the auto chapter, because you won't know what the computer's up to. In this situation, you should edit the auto chapter after you save it. Insert the three characters `V D` where you want the Echo to stop talking, and insert `V A` where you want speech to turn on again.

- ***Stopping an auto chapter before a menu prompt***

As mentioned in Part 2, you can only press control-S to save the auto chapter at a menu prompt. When you want to have an auto chapter that stops at another point in BEX, you can edit the auto chapter and delete some of the final commands, taking care to preserve the last magic <Del> character.

Here's a wacky example. You want to generate a proofreading copy of the text in your Ready chapter, and then compare the printed copy with the text

in the Editor. At the Main Menu, press control-R to start remembering. Edit the Ready chapter, press Tab to insert, then type

```
$$s3$$l2$$ml5$$vf$$rDraft -<Del>-
```

then press control-Q to finish the insert and quit the Editor.

Print the Ready chapter to printer 1, then immediately press to cancel the printing. You're still in remember mode. Edit the Ready chapter again, press control-D control-P to delete the format commands for the draft version, then press control-Q and you're back at the Main Menu.

Press control-S to save the auto chapter, and supply `]` as your name so you can edit it right away in the Ready chapter. Now, locate for ;Q> and delete it. Copy the Ready chapter to the chapter `DRAFT` on your program disk. Whenever you want to create a proof copy, press control-A and type `1DRAFT <CR>` When the printout's done, BEX is ready for you to edit the Ready chapter; your cursor is waiting at character position 0.

- ***Auto chapters and extended disk systems***

Here at Raised Dot Computing, we create braille editions of the registered customer list on a weekly basis. One auto chapter does most of the work; we need to make four minor changes in this auto chapter every time we use it.

This auto chapter kills all the chapters on the disk in drive 45 that end with a particular character. Then it reads the ProDOS textfiles in the disk in drive 44 to BEX chapters on the RAM drive 47, adding the Month and Date to the textfile names to make the source chapter names. Next, it jumps to the Main Menu, and translates all the chapters on drive 47 that end with the last digit in the date, writing the braille chapters on top of the inkprint. Next, it replaces all these chapters using the transformation rules in the `AW-TRANS` chapter on the disk in drive 45. The replaced target chapters are written on drive 45, so we can take the disk downstairs for brailling. The final step is to print the `ALERT` chapter; this chapter just contains ten ;G> characters. When we hear ten beeps, we know that the auto chapter is finished.

In the Editor, the auto chapter looks like this:

```
SK45/*<CR>Y<CR>Y<CR>R44<CR>Y<CR>46.A M-
DD<CR>JR46/D<CR>Y<CR>46.S<CR>45AW-
TRANS<CR><CR>G46/D<CR>Y<CR>45.S<CR>P45ALERT<CR><CR
>SW<CR><Del>
```

We have to modify the asterisk and *M-DD* and *D* characters, so we start out by copying it to the Ready chapter. We locate for the asterisk and replace it with the last digit of the date of the previous braille edition. We locate for *M-DD* and replace it with today's month and date, then locate for *D* and type the last digit of today's date. At the Main Menu, we press control-A. When BEX prompts `Auto chapter:` we respond with `] <CR>` and wander away. In around 15 minutes, we hear ten beeps and we know the information is ready to braille.

8 Input/Output Commands

As we introduced in User Level Section 2, BEX controls four output channels: screen, voice, braille, and printer. At the Master Level, you can issue commands to affect all four channels. You can change the screen size at menus as well as in the Editor. You can activate and deactivate output to any of the four channels. You accomplish these feats with control-B commands. These commands are not available at either the Learner or User Level.

Control-B Command Syntax

You can issue control-B commands at any BEX menu, or, when you have Quit BEX, at the BASIC prompt. These commands are all three keystrokes long. The first character is control-B. The second character is one of four letters that signify which channel you're addressing: V for Voice channel; B for Braille channel; S for Screen channel; and P for Printer channel. The last character depends on which channel you're addressing; details below.

- ***Screen channel***

No matter how you configure, BEX always outputs on the screen channel. How many columns you request for the screen decides the default size. In the Editor, you can change the size with control-S S [letter] where *[letter]* is

one of the ten screen size codes. (Check the Thick Reference Card for the codes to use.) At the Master Level you can also change screen size at menus with control-B S [letter]. Suppose you configure with Echo speech and 80-column screen. A partially sighted person wanders by and wants to know you're doing. At the Main Menu, you press control-S S L to set 20-column screen; now they can see what you're up to.

- ***Printer channel***

Generally, BEX only sends output to the printer channel when you tell BEX to print. Control-B commands let you send text to the printer channel while you're engaged in the computer dialogue. The command is control-B P #, where the number corresponds to the printer number. When you configure printer 1 as a VersaPoint embosser, then entering control-B P 1 sends all the user dialogue to the embosser. You can only have one printer active at any one time. Turn the printer channel off with control-B P D. When you want a hardcopy record of what's on disk, Zip to the Page Menu, activate the printer channel, and use option W - Whole disk catalog.

- ***Voice channel***

The voice channel comes in two flavors. When you have an Echo or SlotBuster, the voice channel is a cooperative effort between BEX and TEXTALKER or SCAT. Alternatively, you can output the voice channel through a serial voice device when you answer the configuration question appropriately. Whether integral or serial, you can turn off the voice channel at menus with control-B V D. Turn it back on again with control-B V A.

When BEX recognizes an Echo or SlotBuster in your Apple, then BEX loads TEXTALKER or SCAT into memory. Even if you don't configure with Echo or SlotBuster output, entering control-B V A turns on speech output. Because serial voice devices must be defined in your configuration, you can't turn them on with control-B V A if you haven't configured them.

Inside the Editor, you can toggle the voice channel off and on. When the voice is active, control-S V turns it off. After you've turned it off, control-S V

turns it back on. You can't turn on an inactive voice channel in the Editor; in other words, the voice must be on to turn it off with control-S V in the Editor.

- ***Braille channel***

An early configuration question asks `Do you have a braille device for all the material going to the screen?` When you answer Y then you have configured the braille channel. After you have, control-B B D deactivates the braille channel, and control-B B A turns it back on again. In the Editor, you can turn off output to the braille channel with control-S B. When you want braille channel output again, entering control-S B a second time does the trick. As with the voice channel, you can only turn the braille channel off in the Editor when it's on to begin with.

9 Changing Your BEX Disk

There are two parts of BEX you can change. You can alter many of the prompts and messages that BEX generates. The text is stored in chapter `MESSAGES` on the Boot disk. As shipped to you, the messages are fairly formal. You can change `Main Menu:` to `What now, hey:` if you like, thereby giving your BEX a beatnik personality.

You can also change the braille translation tables. Both tables are stored as BEX chapters on your Main disk. Chapter `ZQFOR` contains the forward translation table used in grade 2 translation. Chapter `ZQREV` contains the reverse translation table, used in Back translate from grade 2.

Both the MESSAGES chapter and the translator tables use a rigid structure. If you alter this structure, BEX won't work correctly. Of course, you never want to change your Master BEX disk, but you always use a working backup anyway (don't you?) Before you even contemplate making any changes, copy the original chapters to a backup disk. You can then recover from errors by copying the originals back to your working copy of the BEX disk.

Part 1: The MESSAGES Chapter

On the boot side of BEX is a three page chapter called MESSAGES. This chapter contains the text of many of the prompts and messages produced by BEX. BEX loads one of these pages into memory when you supply a configuration name. Which page it loads depends on the level of the configuration: Page 1 is for the Learner Level, page 2 is for the User Level, and page 3 is for the Master Level. Actually, BEX doesn't load the page based on the BEX page number; it uses the page filename. BEX loads `MESSAGES.A` for the Learner Level, `MESSAGES.B` for the User Level, and `MESSAGES.C` for the Master Level. When you edit this chapter, make sure that the page file extension letters follow this pattern.

The MESSAGES chapter is the mechanism by which the prompts get shorter as you advance through the levels of BEX. If you are familiar with BEX, you can shorten the prompts to the point of being absolutely cryptic. You are limited by your imagination and the size of the page. No page of the MESSAGES chapter may exceed 3840 characters. After we discuss how you change the MESSAGES chapter, we supply some reasons you might want to do it.

The Structure of the MESSAGES Chapter

The delimiter between text sections is the <Del> character. When BEX uses text from the MESSAGES chapter, it locates it by counting <Del> characters. For example, when you press X at the Main Menu, BEX outputs the 23rd message. It counts 23 <Del> characters, then outputs all the text up to the 24th <Del> character. As shipped to you, the 23rd message is `Illegal option <CR>` The ;G> is the ASCII *bell* character, so the Apple beeps. BEX outputs the words *Illegal option* on all four channels. The <CR> at the end means that there's a new line on the screen and printer channels.

If you accidentally add or delete a <Del> character, you can make BEX unusable, because most of the time it would be giving you the wrong prompts. So do be *very* careful. Please make a backup copy of the original MESSAGES chapter before you even think about modifying it.

When you think you've correctly modified the MESSAGES chapter, copy the new text onto the boot side with the chapter name MESSAGES. Make sure

that the page files are in alphabetical order. Any changes that you make show up after you boot BEX with the altered MESSAGES chapter.

The final characters in each BEX page are the vocabulary BEX uses when arrowing in the Editor. As a test of your success, after you reboot with a modified MESSAGES chapter, move to all four menus, and list the options. Then use the arrow keys in the Editor. When everything works correctly, then you know you've done it right.

- ***Special characters in MESSAGES***

There are three characters in the messages chapter that control when a <CR> is sent to the voice and screen channels. Any <CR> in the MESSAGES chapter is executed for all channels. Any caret ^ instructs BEX to send a <CR> *only* to the voice channel. Some voice devices won't speak until they get a <CR>, so almost every prompt ends with caret. If you remove the caret, then you won't get immediate voice output with DECtalk or Votrax.

The tilde ~ character marks a <CR> that only appears in 80-column or 40-column non-HI-RES screen modes. When you use any large print screen mode, the tilde in the MESSAGES chapter has no effect. In addition to the ;G> character that makes the Apple beep, messages that supply defaults include a ;H> character. An example is message number 36, which asks if you want to start a new chapter when editing. At the Master Level, this message is `Start new? Y ;H>` is the same as left arrow, so this control character moves the cursor left on top of the letter *Y*.

Order of Prompts in MESSAGES Chapter

- 1. Main Menu options list
- 2. Second Menu options list
- 3. Which printer choices
- 4. Choices at the generic chapter prompt
- 5. Target naming method choices
- 6. Update date
- 7. Insert program disk
- 8. Insert data disk
- 9. Program segment not loaded

- 10. Chapter not found
- 11. Information after you do `RUN 999`
- 12. Starting Menu options list
- 13. Page Menu options list
- 14. Starting Menu prompt
- 15. Main Menu prompt
- 16. Second Menu prompt
- 17. Page Menu prompt
- 18. Generic chapter prompt; at Learner Level, this is `Drive number or chapter name:`
- 19. Target chapter prompt
- 20. Target chapter naming method prompt
- 21. Warning when you initialize a disk
- 22. Insert program disk and hit any key
- 23. Illegal option
- 24. Response when you press P at Main Menu
- 25. No choice selected (only used at learner level)
- 26. Transformation chapter not found
- 27. Save transformation chapter
- 28. Chapter number prompt when choosing from numbered list
- 29. Disk error, cancel remember mode
- 30. Error, identical file names
- 31. Disk read error
- 32. Chapters located (when scanning for chapters)
- 33. No chapters located
- 34. Textfiles located (when scanning for textfiles)
- 35. No textfiles located
- 36. Response when you press E at Main Menu
- 37. Use transformation chapter
- 38. Auto chapter
- 39. Textfile prompt during Read or Kill textfile
- 40. Target textfile prompt
- 41. VersaBraille transfer error
- 42. Unused
- 43. Unused
- 44. Disk cannot be read
- 45. Do you want to start a new chapter?
- 46. Bad chapter name
- 47. Disk error in Editor
- 48. Response when you press F at Second Menu

- 49. Chapter prompt when you can not scan a disk
- 50. Help messages for prompt 49

The remainder of items are the vocabulary used for voice output of arrowing in the Editor.

Here are some ideas for modifying the MESSAGES chapter. When you use 10-column screen mode, you can shorten all the prompts so they fit in one line. You could use Replace characters to change every appearance of the word *chapter* to uppercase *C,* saving you six characters per prompt. Our programmers prefer that prompts end with a colon or question mark, but if you generally set your Echo for most punctuation, you might find the colon intrusive. Again, use Replace characters to simply delete them.

If you feel that the computer should adopt a more servile attitude, you can change the menu prompts to be much more polite. You could replace `Main:` with `My wish is your command, Milady` You could change `Target chapter:` to `Phasers locked on target!`

A more practical example concerns the Page Menu. It's the only Master Level prompt that's two words long. We wanted to distinguish between the menu prompt and the frequent `Page:` prompts when you need to enter a BEX page number. You could shorten the menu prompt to simply `Page:` if you added Echo commands. Replace the 17th entry in the MESSAGES chapter with:

```
31 P Page:^ 21 P
```

and then reboot. The high-pitched `Page:` signals the menu prompt, while the low-pitched `Page:` means a page number.

One last word of warning. All Levels of the BEX manual show many examples of user dialogue. When you change the MESSAGES chapter, then the program no longer matches what's in the manual. This could prove very confusing for beginners, so think before you edit.

Part 2: ZQFOR, The Braille Translation Table

The grade 2 braille translator works by applying the rules in the chapter `ZQFOR` to your text. The two page chapter ZQFOR is located on the Main side of your BEX program. ZQFOR rules are in a particular format that drives the translation process. Changing the table will change the way the translator works. You might want to change the table to improve the translator or to add your own, nonstandard rules to it. We would appreciate receiving information about any problems you find with the existing Grade 2 translator.

Warning! Changing ZQFOR may result in program crashes. Never contemplate making translator changes without making a backup copy. To make a backup copy, copy chapter ZQFOR to a data disk.

The ZQFOR table is broken up into 29 sections. Each section is separated from the next section by two consecutive <CR>s. The first section of the table holds the translation rules for the space character. The second through 27th sections of the table hold the translation rules for the letters, *a* through *z.* The 28th section holds the translation rules for punctuation characters. The last section holds the rules for returning from a special translation mode. Numbers are translated internally to the program. Control characters pass through without modification.

Each rule takes up a line of text which is broken into one, two, or three parts. The first part is the matching string. Each letter of the text is matched against the string of all of the rules for the letter until one string matches or the end of the table for the letter is reached. The matching string is terminated by one of three letters:

- E - Change the portion matched to the string that follows
- S - Copy the current character to the output and resume at next letter. In effect an *S* means stop the search through this table. The entry `haddS` - prevents the use of the *dd* contraction in favor of the *had* contraction in the word *haddock.*
- X - Copy the portion that matched to the output. The X code is just like the E code, it is a shorthand since the *change to* string does not have to be in the table.

There are other letters with a more specialized function--contact us if you need an explanation. Capital letters in matching strings direct the translator

to apply a special rule to the text rather than matching rule against text one for one. Some of the matching string rules are:

- B - matches blank or control character
- L - matches upper or lower case letter
- N - matches number
- P - matches punctuation.
- Q - matches blank, control character or punctuation
- R - matches if blank, apostrophe *s*, or dash followed by a letter or number.
- T - matches if letter preceded by a number.
- V - matches any vowel.

A capital letter at the beginning of a match string is not compared directly with the text. Rather it is matched with a one character buffer that holds the computer's version of what was the last character it dealt with. This buffer, which we will call the *before character* can be modified in midstream. For instance, for the word *fed* when the *f* is matched, a space is the before character, then when the *ed* is matched, the *f* is the before character. Finally, while the ending space is matched, the *e* in *ed* is the before character. If a Z appears at the end of a replacement string, then the character after the Z becomes the before character. For instance, the rule `N,NE1Z1` changes a comma in between numbers (like in 100,000) to the computer braille for comma, then changes the before character from a comma to a number to keep the translator from inserting an unneeded number sign before the *000.* There are special meanings for certain characters after the Z. These are:

- ZB - go into computer braille
- ZC - go back to standard translation (from computer braille or English mode)
- ZE - English mode, suppress use of the capital sign (dot 6)
- ZF - set a flag indicating that it is okay to swallow a space (for example *for the* loses a space)
- ZW - leave in computer braille until a space or the end of a page (used for format commands)

The last detail of the rule application process is the setting of boundaries of the portion of the string that is replaced. The only part that is replaced is the portion of the text that matches the part of the rule before the first capital (other than the before matching capital). For example, the rule, `bVVX` copies

the *b* to the output, but not the two vowels afterward that matched. The rule `QbeLLLE2` replaces the *be* with the computer braille `2` (dropped b).

The best guide to changing the table, after the above, is the translation table itself. It provides a wealth of instructive examples.

Part 3: ZQREV, The Back Translation Table

The back translator turns a grade 2 braille chapter into a printable chapter. It performs the reverse function of the braille translator. As with the forwards translator, you should always make a backup copy of the `ZQREV` translation table chapter before making any changes. Failure to make a backup copy can cause frustrating circumstances.

The back translator driving table works in the same basic way that the translator table for the forward translator works. The back translator table is responsible for the entire translation process.

The back translator table has 65 different sections. Each section is marked by two <CR>s. The table sections are in the ASCII character order. The table starts with ten <CR>s. Then comes the section for space, followed by the section for `!` the *the* sign. After the table for `@` comes the 26 letters, followed by the section for getting out of computer braille.

ZQREV also has different capital letter special rules for the matching string. These rules are:

- B - matches blank or control code
- L - matches letter
- M - matches blank, punctuation, a dash or matches if next letter is capital
- N - matches number
- P - matches punctuation
- Q - matches blank, control code or punctuation

It is not trivial for the back translator program to determine what category a character is in. For example, virtually every braille cell (every cell except space and dot 3) can be turned into a letter in some circumstance.