Getting started with Tutorial 6
The Phone List

GETTING STARTED WITH TUTORIAL 6

Once again, you should have your phone list handy. If by this time you're running low on phone numbers to input, find all those numbers you jotted on napkin scraps, have the phone book handy, or track down the addresses and phone numbers of all your creditors. Our goal is to create a phone list file containing every conceivable number you use during ordinary work.

If you're wondering when on earth you're going to start USING this file, don't fret. Tutorial 9, coming up at the end of this week, introduces Grep and Sort. Offered on the "Mastering OS-9" disk, these two commands make finding phone numbers fast and easy.

What we'll cover

Compared to many newer computers, and to most older computers such as the IBM PX, XT and even AT, your CoCo uses memory efficiently. This memory efficiency makes an even greater bargain of the 512K CoCo 3. Even with 512K, memory is a precious resource that must be carefully handled.

In this tutorial we offer two ways to conserve memory. One technique involves combining several commands into a larger command file using merge and attr. We'll check our work by actually loading the new command file to determine how it fits into memory. Another technique removes any command from memory not currently in use with the unlink command.

As the tutorial proceeds you monitor available memory with the familiar mfree command. You also keep tabs on which commands occupy memory with the mdir command.

You can find the size of commands with the dir e. Since dir e reports all numbers in hexadecimal be prepared for a little work with hex. (We introduced hexadecimal notation in the essays you should have just completed.) Keep convert handy. You should find Tutorial 6 easy going.

Only carry what you need.
Commands you will use - **unlink**

Under OS-9 Level 1, the first version of OS-9 for the CoCo, memory was extremely precious. Users were forced to plan their use of the computers memory very carefully. One technique used was to load commands into memory only when necessary. After the usefulness of a command was over it could be removed from the system with the unlink command.

For example, suppose a particular job requires repeated use of the format command. The Level 1 user types:

OS9: load format <ENTER>

After formatting all the disks and ramdisks on hand the user removes the command from system memory with:

OS9: unlink format <ENTER>

How linking works

Although OS-9 Level 2 offers far more memory space than Level 1, this "load-unlink" process can be a big help : especially since commands loaded under Level 2 usually take up more memory than they did under Level 1.

Most commands are loaded into memory not by you but by OS-9 with its version of the load command. OS-9 then gives the command a link count of one. While the command is being used : while it is linked to the system : it is available for other processes to use also. If any processes access the command, OS-9 continues to boost the command's link count one time for each process using it.

For example, imagine you have several applications running at once. Each application, running as an OS-9 process, may use OS-9 commands as part of its design. As each process finishes using a command in memory, OS-9 (invisibly to you) decreases the link count by one with the Kernel's version of the unlink command. When every process is finished with the command in memory, its link count becomes zero and OS-9 removes it from the system. The next time you call the command at the "OS9:" prompt, the whole procedure repeats : including the initial disk seek to find the command.

We have shown before that disk accesses are much slower than accesses to memory. To avoid this constant reseeking you can boost the link count to equal one more than the total number of processes currently using the command. After you boost the link count, even when all processes are finished using your in-memory command, it still has a link count of one and thus remains in memory.

The easiest way to boost the link count is to use the load command. We outlined the procedure above. We also mentioned how to remove the command from memory with the unlink command. This command decreases the link count by one each time it is called at the "OS9:"prompt

Why linking may not work

Using the unlink command will not necessarily remove the command from memory. The reason lies with how OS-9 makes in-memory commands available to your computer's programs.

Each time you load a command the link count goes up. Suppose you have just run three procedure files each of which included "load makdir" as one of its lines. Unless each procedure file also includes "unlink makdir" at the end, OS-9 increases the link count altogether by three. A SINGLE use of the unlink command would be insufficient to remove makdir from memory.

To remove <command. name> from memory it may he necessary to unlink <command. name> over and over until the command disappears. Once you remove the command any additional use of unlink <command. name> returns ERROR #221 - Module Not Found. This is, of course, not really an "error" since removing the module from memory is exactly what you have in mind!

Don't over unlink!

The "load-unlink" process normally does the trick, allowing you the luxury of having a valuable command in memory while you need it and the luxury of getting back the memory it occupied when you finish using it.

You may run into trouble one day when you repeatedly unlink a command or other memory module currently in use by another process. If you succeed in removing a command from memory while it is in use by a program or procedure file you may create havoc. Be cautious when unlinking a command which may be needed by other active processes (You can check which processes are active with the procs command).

The 8K block size and you

The next memory-saving technique requires an understanding of how the CoCo 3 allocates memory. The hardware inside your CoCo 3 allows it to access a great deal of memory : up to 512K (2 megabytes with aftermarket upgrades). But there is a compromise. In general, the smallest amount of memory OS-9 can allocate is 8K. The reason for the compromise is beyond the scope of this book and is irrelevant to most users.

Keep in mind these facts:
* The CoCo 3 can "look at" only 64K of memory at once. Which 64K out of the total 512K it "sees" depends on which 64K is presented to it by the Memory Management Unit (MMU : part of the GIME chip in the CoCo).

* The MMU presents that 64K memory's blocks of 8K each.

* Therefore, the total 512K (or more) of your CoCo3 is ultimately organized in 8K blocks.

When you load a command, it occupies its own 8K block. The size of many commands is tiny in comparison. For instance, loading the makdir command : only 34 bytes in actual size : causes a total waste of most of that 8K block of memory!

Commands you will use - merge

You can conserve memory by merging several commands into a larger "command", saving it to disk, and using that larger "command" instead of any of the smaller commands it contains.

An excellent instance of this is the Shell. When the Shell is loaded from disk during bootup nineteen more commands enter along with it. Tandy merged these with Shell to avoid wasting 6,500 bytes in the 8K block Shell occupies. Most of these are commonly used commands so system responsiveness is improved.

Here is -a list of commands merged into the Shell under Level 2:

Module	Size in decimal notation
Shell	1532 bytes
Copy	743 bytes
Date	253 bytes
DelniZ	134 bytes
Del	165 bytes
Dir	875 bytes
Display	132 bytes
Echo	34 bytes
Iniz	122 bytes
Link	44 bytes
List	79 bytes
Load	36 bytes
MDir	797 bytes
Merge	104 bytes
Mfree	491 bytes
Procs	821 bytes
Renaue	285 bytes
SetiMe	280 bytes
Tmode	769 bytes
Unlink	45 bytes

That's a command-packed Shell! There is still a little more room too, since these sizes add up to only #7721 bytes.

Because your Startup file links the Shell these commands remain in memory at all times, instantly ready for use.

Before you merge commands, decide which ones you want together. The aim is to collect in one large, executable file a handful of commands commonly used in the same work session.

One common work session is disk "house cleaning". Disk house cleaning involves copying and deleting files, making and deleting directories, and using the dsave command. At the end of such a session your disks will be efficiently organized and free of files you no longer need (We cover dsave in Tutorials 7 and 8). Since this work session requires repetitive use of these commands you might deem it wise to merge and load any commands not currently in memory.

You can find the names of memory-resident commands with the mdir command. The listing mdir provides shows copy and del are accounted for in memory. That leaves dsave, deldir, and makdir to be loaded by hand. It makes sense to merge these together to save memory. The resulting file can be used for any future house-cleaning sessions. Let's examine the size of these commands to see if they can squeezed into an 8K block:

Module	Size in decimal notation
Dsave	6680 bytes
Deldir	636 bytes
Makdir	34 bytes
Attr	645 bytes
TOTAL SIZE	7995 bytes... Perfect!

Merging files and changing attributes

Now that the size research is done, how would we merge these commands together? First, think of a name under which to merge them, say, Clean.up. Then change your current data directory (chd) to /d0/CMDS. Why data directory? Because merge joins files in the current data directory. Then use the merge command. In this instance, you would type:

OS9: chd /d0/cmds <ENTER>

OS9: merge dsave deldir makdir attr > Clean.up <ENTER>

Clean.up will be saved in the current data directory. Merge creates Clean.up as a data file and not an executable file. This means that Clean.up is useless to us as a command until we tell OS-9 that it's executable. We do this with the attr command.

You should recall from the introductory reading that, in general,

OS9: attr <filename> <ENTER>

lists <f ilename>'s file attributes to your screen. These file attributes tell whether the file is a directory, shareable, executable,
and by whom. In this case, we can expect

OS9: attr clean.up <ENTER>

Our report will read: " ----r-wr ", which is bad news to us since the letter "e", signaling an executable file, is omitted from the report. You can set : turn on : that attribute with:

OS9: attr clean.up e pe <ENTER>

This sets the executable attribute for both the super-user and the public. Once attr has finished, it displays the new attributes to your
screen: " --e-rewr "

You follow a procedure similar to the one outlined above when you begin the tutorial. For now, just remember that when you load clean.up into memory you speed up disk "house-cleaning" without taking up unnecessary memory. Creating Clean.up will save you time and 24,000 bytes (24K) of memory!

Tutorial 6
Step 1: Loading and unlinking commands

TUTORIAL 6

STEP 1: LOADING AND UNLINKING COMMANDS

1.1 Replace the write-protect tab on "Custom Boot # 1 Backup". Have your backup of the Basic09/Config disk handy.

1.2 Boot up OS-9. Place your "Mastering OS-9 Backup" disk in drive /d1. Once the boot proces is complete, type:

OS9: dir x <ENTER>

Recall that the x option of the dir command lists the contents of the current execution directory. At boot-up, floppy-disk based systems always have /d0/CMDS set as the current execution directory.

1.3 Type: **OS9: dir /d1/CMDS.sO9 <ENTER>**
These are the commands we provide on the Mastering OS-9 disk.

1.4 Now type: **OS9: mdir <ENTER>**
Once again, these are the modules and commands resident in memory on bootup. The commands you see were all merged into the Shell and linked to your system in the Startup file.

1.5 Type: **OS9: tmode . 1 pause <ENTER>**
This will stop the screen from scrolling if it fills up with text. Now type: **OS9: list startup <ENTER>**
find the line which says: link Shell

This step links the Shell and nineteen other commands into memory. If necesary, press either <BREAK> or <SPACE> to return to the "OS9:" prompt.

1.6 Now we're going to simulate a typical Level 2 session by using four windows, each with its own purpose. In the first window we'll start up the BasicO9 programming environment. Then we'll open two windows to edit large files such as programs or reports. In the last window we'll imagine doing a disk "house-cleaning" session like the one mentioned in the introductory reading : but without merging commands at first. Our goal is to consume as much memory as possible. Type:

OS9: chd /d 1 <ENTER>
OS9: make 1 2 <ENTER>

This procedure file, in the root directory of the Mastering OS-9 disk, starts two windows in addition to the ones you initialized in Startup.

1.7 Remove the Mastering OS-9 disk from drive /d1. Place "BasicO9/Config Backup" in drive /d1. Now type:

OS9: mfree <ENTER>

Remember this number.

1.8 Now type the following:

OS9: chx /d1/cmds <ENTER>
OS9: load basicO9 <ENTER>
OS9: load runb <ENTER>
OS9: mfree <ENTER>

Loading BasicO9 and RunB uses up 40K or 5 blocks of memory.

1.9 Type: **OS9: basicO9 #32k <ENTER>**

This starts BasicO9 and provides it 32K of memory for program storage. If you were involved in an actual programming session with BasicO9 and RunB you could eat up a lot of memory.

STEP 2: STARTING THE EDIT SESSIONS

Step 2: Editing startup

2.1 Remove the Basic09/Config disk from drive /d1. Replace it with the "Mastering OS-9 Backup" disk.

2.2 Press the <CLEAR> key until a screen with two windows appears. Make sure you see the cursor on the first window. Now type the following commands:

OS9: chd /d1/TUT6.EDIT.FILES <ENTER>
OS9: tmode .1 pause <ENTER>
OS9: edit #32k File 1 <ENTER>
E: - * <ENTER>
E: 1 * <ENTER>
Press the <CLEAR> key.
OS9: chd /d1/TUT6.EDIT.FILES/TWO.SEV <ENTER>

You must use the chd command above because each window behaves like a separate computer, requiring "orientation" to its place in the directory system. If you don't use these commands, a window inherits the current directories of the window which created it. For example,since/d0 and /d0/CMDS are the current data and execution directory when you boot OS-9, the windows created by Startup all inherit those directories as data and execution directories.

Riddle: When you created the two windows in Step 1.6 above, what current execution and data directories did OS-9 give them?

2.3 The previous step loaded Edit into memory from /d0/CMDS. The light on the front of drive /d0 came on and drive /d0 whirred until the command was in memory. Then the Shell loaded in File1 from drive/d1, which also lit up and whirred. Notice this time, however, only the light on drive /d1 will come on. The Shell does not need to get edit off of drive /d0 because it is in memory. Edit's
link count is bumped up one:

OS9: edit #32k File2 <ENTER>
E: l * <ENTER>

Step 3: Clean-up

STEP 3: USING THE LAST WINDOW FOR CLEAN-UP

3.1 Imagine you are a full-fledged OS-9 guru whose purpose is to patch up mistakes in modules with modpatch, xmode, and ident (all of which help make informed, accurate changes). After all these changes you intend to do disk clean-up as we described in the introductory reading. This requires dsave, makdir, deldir, and free
to be in memory. You'll also need copy, already present in memory, having been loaded with the Shell during bootup. Let's see just how much memorywe have to work with by typing:

OS9: mfree <ENTER>

Remember the number reported.

3.2 Now type these commands:

OS9: load attr modpatch xmode ident <ENTER>
OS9: load dsave makdir deldir free <ENTER>
OS9: mfree <ENTER>

There goes your memory! "But", you say, "there's more memory left than what I had altogether on my CoCo 2!" True, but much of the fun and productivity of Level 2 comes from having many programs in memory, including your word processor, spreadsheet, and telecommunications package. WizPro, a shareware telecommunications package by Bill Brady has 128K of code! Imagine loading in these applications on top of the commands you already loaded! In short, it's difficult but not impossible to eat up all the memory in your 512K CoCo 3.

Step 4: Unlinking

STEP 4: UNLINKING

4.1 Press the <CLEAR> key to return to the BasicO9 screen. At the "B:" prompt, type: **B: bye <ENTER>**
Press the <CLEAR> key to proceed to the first Edit screen. we'll get out of edit by typing: **E: q <ENTER>**
Press the <CLEAR> key again and let's exit the other edit window: **E: q <ENTER>**
Press the <CLEAR> key one more time.

4.2 Now we need to know just what commands are in memory again, and get rid of some we nolonger need. Type the following , watching your screen after each mdir command:

OS9: mdir <ENTER>
OS9: unlink modpatch xmode ident <ENTER>
OS9:unlink dsave makdir deldir free <ENTER>
OS9: mdir <ENTER>

The commands listed after unlink should now be removed from the
system. If some remain, use the unlink command until each has disappeared. You can either 1) checkyour work with mdir after each uUnlink or 2) unlink each until you receive ERROR 221, which means OS-9 couldn't find the module in memory.

4.3 Type: **OS9: unlink basicO9 <ENTER>**
As mentioned in the introductory reading, under some circumstances, repeated unlinking can cause problems. If you were using a Basic09 application in another window, repeatedly unlinking Runb or Basic09 will probably cause that application to crash, since it relied on having these in memory to run.

4.4 Let's see what is now in memory: **OS9: mdir <ENTER>**
Notice that edit is not in memory. As soon as you quit the first edit
session OS-9 unlinked edit, reducing its link count by one. As soon as you quit the second edit session OS-9 unlinked the command again. Since no other processes used edit, it's link count was zero and OS-9 removed it from the system. OS-9 also took care of de-allocating the memory you asked for in Steps 2.2 and 2.3 (the edit buffers).

STEP 5: MERGING COMMANDS TOGETHER

5.1 Have a pencil handy. The f ollowing steps make frequent use of the attr and merge commands. Attr is already in memory, so we only need to load merge: **OS9: load merge <ENTER>**

5.2. On a full 80-column screen, type **OS9: dir x e <ENTER>** If your current window overflows, use tmode to pause it (OS9: tmode .1 pause). Then type: **OS9: dir x e <ENTER>** again. Write down the sizes of dsave, makdir, deldir, and attr. These are listed in the column marked bytecount. They will be hexadecimal numbers. When you're done you should have on your paper:

Bytecount	Name
285	attr
118	dsave
22	makdir
27C	deldir

5.3 To save you the hassle of converting hexadecimal numbers to decimal, we have provided convert, a simple program in BasicO9 which provides hex, decimal, and binary versions of the number you type in. The number you enter can be in any of the three forms. To run convert:

1) Runb should be in memory. Check with mdir. If it is, go to part 4 below. If not, load it from /d0/CMDS (where it should also be) and go to step 4. If absent from /d0/CMDS, insert the Config/ BasicO9 Backup disk in drive /d0.

2) Type: **OS9: load /d0/cmds/runb <ENTER>**
and wait for RunB to load.

3) Remove Config/Basic09 Backup disk from drive /d0 and replace it with the system master backup.

4) Type: **OS9: load /d1/cmds.sO9/convert <ENTER>**

5) Type: **OS9: convert <ENTER>**

6) Follow the prompts, making sure to precede each of the following hex numbers with the "$" sign.

(for attr) $285
(for dsave) $1A18
(for makdir)$22
(for deldir) $27C

5.4 Copy down the decimal equivalent of each module. Add them up. The fundamental properties of addition haven't changed since we performed this sum in the introduction, so you should get 7995 bytes. Since 7995 bytes is fewer than the 8191 byte block, these commands may be merged and loaded without memory waste.

Remove the write-protect tab from your backup system master if one is present. Type the following commands:

OS9: chd /d0/cmds <ENTER>
OS9: merge attr dsave makdir deldir > clean.up <ENTER>

Your drive /d0 will spin for a while as OS-9 works.

5.5 Now we need to make the new file executable. We also want to know how much memory is free and how much is used:

OS9: attr clean.up pe e <ENTER>
OS9: mfree <ENTER>
OS9: load clean.up <ENTER>
OS9: mfree <ENTER>

Notice that only 8K of memory is gone. If you had loaded these commands individually you would have lost 32K of memory.

STEP 6: MERGING OTHER COMMANDS TOGETHER

Step 6: Merging other commands together

6.1 With the Mastering OS-9 disk in drive /d1, type the following:

OS9: chd /d1/cmds.s09 <ENTER>
OS9: dir e <ENTER>

You should see something like:

```
  Directory of . 18:41:36
Owner  Last modified  Attributes  Sector  Bytecount  Name
-----  -------------  ----------  ------  ---------  ----------
   0   88/08/24 1820  -e-rer        1C1         FD  sort
   0   88/08/24 1820  -e-rewr       1C3         FF  d
   0   88/08/24 1820  -e-rer        1CB        3B2  grep
   0   88/08/26 1338  -e-rer        191        2DE  copy
   0   88/08/24 1820  -e-rer        214        60F  convert
   0   88/08/26 1335  -e-rewr       19B         26  cls
   0   88/08/26 1336  -e-rewr       19C        1FE  count
   0   88/08/28 1717  -e-rer        10B         CA  uniq
```

With the Convert utility in another window, convert the hex sizes listed in the Bytecount column into decimal numbers. Do they all fit into an 8K block? Merge as many as you can into a file. Put copy as the first command listed after the merge command (refer back to 5.4). Name the new file "new.copy". Set the execution attributes (see 5.5). All done? That wasn't hard : just a little tedious.

6.2 After reading the descriptions of these commands (see Appendix F):
1) Rename the old copy command on your backup system master to copy.old.
2) Copy new.copy over to the CMDS directory on your backup system master, giving it the name Copy.
3) Insert a line in your Startup file which loads your new copy command on startup.

6.3 I'll demonstrate the above three steps. Put the system master backup in drive /d0 and the Mastering OS-9 backup in drive /d1. Now type the following:

OS9: chd /d1/cmds.sO9 <ENTER>
OS9: rename /d0/cmds/copy copy.old <ENTER>
OS9: copy new.copy /d0/cmds/copy <ENTER>
OS9: chd /d0 <ENTER>
OS9: edit startup <ENTER>
E: + *
E: <SPACE> load copy
E: q

Step 7: Phone list again!

STEP 7: THE USUAL LAST REQUEST

Get out the Phone List and other files disk and use the edit command to add more phone numbers to the list. Remember you can ask for more memory if you need it by calling the editor like so: **OS9: edit #32k phone_lf <ENTER>**
Questions? Refer to Tutorial 4.

Note: Just to check your work in Step 5.3, 25 hex = 645 decimal, 1A18 hex = 6680 decimal, 22 hex = 34 decimal, and 27C hex = 636 decimal. The total size of the files in Step 6.1 is 4489 bytes, well under the limit. If you wish, you can add more OS-9 commands (such as format, which takes up 3134 bytes, or backup, wcreate, and free, which take up less than 3000 bytes). Or add other utilities from third party vendors.