Working with OS-9 — A Comprehensive Manual

Build a beacon station on your CoCo. Follow a report from a shell command through a process, a memory map and a device driver. Then write the program that sends it.



The running CoCo shows OS-9 assembling and identifying the station's beacon module.

Brian K. Holman

Book 11 · VCC Plus 3.7.1

© Brian K. Holman 2026

Contents

Which program heard me? 3
Give the shell a job 5
Where did the output go? 7
A file has a name and an owner 9
Follow a filename onto the disk 12
Recover the right copy 15
Catch a process while it sleeps 17
Where does a program live? 19
One address, two views 21
Open a device by name 24
Who owns this terminal? 26
Put the night watch on duty 28
Assemble a program on the CoCo 31
Deliver the beacon report 34
Find a command 36
Device and storage atlas 41
System calls and errors 43
Choose a language and a delivery format 46
Find your way back into the system 49
Publication and artwork notices 51

Which program heard me?

Chapter 1 of 14

Your station needs a log. At the keyboard you can name a disk, ask the time, or start a second command interpreter. The first puzzle is who hears those keystrokes. On this CoCo the answer changes twice between power-on and the first command.

Boot your practice station

In **Machine → Configuration Profiles…**, select **NitrOS-9 L2 V3.3.0**, choose **Create Fresh Starter Media…** and name the personal copy **Beacon Station**. Load that copy, confirm the restart and choose **Run**. Use the same working images for this book. If you are returning from the quick start, you can also use its personal practice set if the new names are unused. The quick start shows the controls.

The prepared profile enters the boot command at RGB-DOS. The CoCo boots the system on HD0 and eventually starts **Shell+** at the terminal. Click the CoCo display. Wait for the prompt ending in **/DD:**. The text before the path identifies the terminal and shell process, such as **{Term|02}**. If you see the BASIC **OK** prompt, follow the return trip to OS-9.

Open **Media → Virtual Hard Disks…**. Identify the image files assigned to HD0 and HD1. The guest uses **/dd** for its default system disk. On this profile **/h0** reaches that same disk. **/h1** holds your practice work. Close the drive window and click the CoCo display again.

Read the system's morning instructions

At Shell+ enter:

```
list /dd/startup
```

You are reading a file already run during boot. Near the top, two echo commands explain the welcome banner. The file keeps the shell resident, loads a set of utilities, and loads a networking module. Farther down it creates windows and starts shells on window devices. Seeing a networking module in this listing does not grant a guest permission to use a Mac shell. That decision belongs to VCC Plus.

The kernel and I/O modules must already be running for that file to be readable. The boot code brought in the operating system, including the modules in **OS9Boot**. The system startup process arranges the initial environment. The shell can then execute the readable startup procedure. Changing a startup command and rebuilding the set of boot modules act at different stages. Keep the system image intact during these exercises. Later chapters let you inspect modules while the station runs.

Predict two search paths

Make a new practice directory on your private HD1 and enter it.

```
makdir /h1/station
chd /h1/station
pwd
pxd
```

pwd returned **/H1/station**, the current data directory. **pxd** returned **/DD/CMDS**, the current execution directory. If the directory already exists, inspect its contents before reusing it. A new copy of the practice media gives you fresh names.

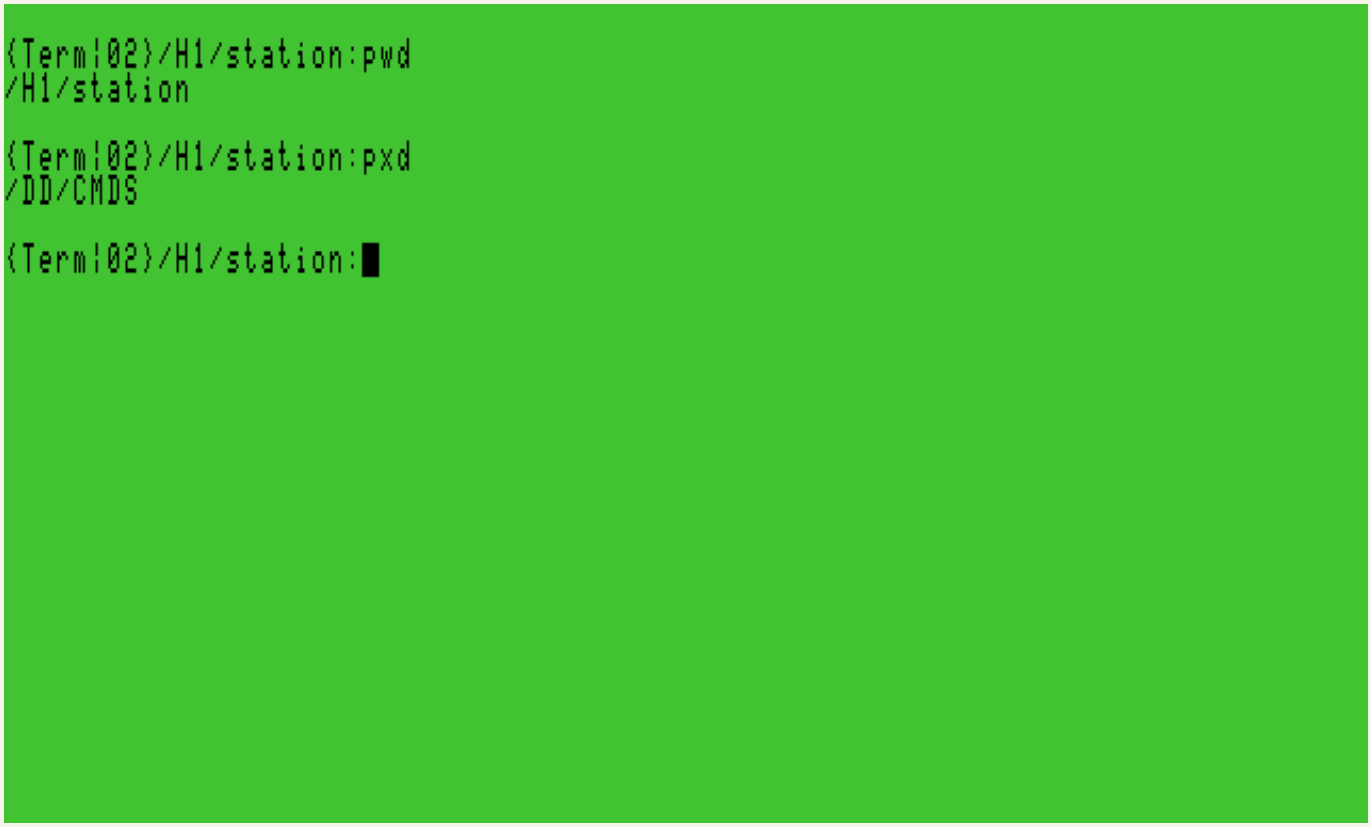


Figure 1: The guest prints /H1/station for pwd and /DD/CMDS for pxd beneath the same Shell+ prompt.

When you ask a utility to open an ordinary relative filename, the data directory supplies the starting point. When you type a program name, Shell+ can find a resident module or search for executable material using its command search rules. The execution directory supplies the usual location of disk commands. A leading slash starts a path at a named device. You can therefore stand in your station directory, run **list** from the command environment and read **/dd/startup** from the system disk.

Prediction. You will shortly type **build report**. Which disk will receive the two-line file? The answer is HD1 because **report** is a relative data filename and the current data directory is **/H1/station**. The shell's ability to find **build** is a different lookup.

Give the shell a job

Chapter 2 of 14

A station operator can remember two commands. A relief operator arriving tomorrow would appreciate finding them already written down. Give the next shift a file that the shell itself can read.

Write and inspect a procedure

Start in **/H1/station**, the data directory created in chapter 1. Enter **build report**. The **?** belongs to Build. Type each of the following lines, pressing Return after each. At the next empty **?** press Return once more. That blank reply ends entry.

```
echo beacon online
pwd
```

Now enter the following at Shell+:

```
list report
shell report
```

list displays the two command lines exactly as stored. **shell report** interprets those lines and displays **beacon online** followed by **/H1/station**. The first word on the outer command line names a program. The word **report** is an argument to it. Inside the file, **echo** receives the words **beacon online**. The second command, **pwd**, uses the directory inherited from the calling environment.

Notice what the procedure leaves behind. You are back at the prompt that started it, still standing in the station directory. A child shell can change its own working directory while running a job. Its parent retains its own directory when the child exits.

See which punctuation the shell handles

Enter these two lines separately:

```
(echo north; echo south)
echo "two words"
```

The first prints **north**, then **south**. The semicolon separates commands that run in sequence. The parentheses group that sequence. The second line prints **"two words"**, complete with quotes. The quotes survive in the parameter text seen by this version of Echo. This matters to a reader coming from a Mac shell, where the output of the same-looking command can differ.

At the OS level, the new program receives a parameter area containing text. Each utility has rules for interpreting that text. Consider the different jobs of **shell report**, **list report** and **echo report**. The same word can name a procedure, name a file to display, or simply be part of a message. Use each utility’s own syntax when an argument can contain a space.

Let the child read until the file ends

Try another way to supply those two lines:

```
shell <report
```

This time an interactive copy of Shell+ receives the file on its standard input. In the practice session it printed its greeting, ran both commands, displayed **eof**, and returned to the original shell. The terminal and process number in the child's prompt help identify the change. The child stops because the file has no more input. Nothing in the file needs to type an exit command.

Programs also return a completion status. A child can finish successfully or give its caller an error number. Output text and completion status are separate ways to report what happened. In the next chapter a failing child's message will travel down a path you choose. Later you will write a module that chooses its own exit status.

Where did the output go?

Chapter 3 of 14

Two operators want different things from the same report. One wants to see a single matching line. The other wants both a display and a saved copy. Neither should have to rewrite the program producing the text.

Connect paths zero, one and two

A path number is a small handle valid in the process using it. A newly started command ordinarily inherits three paths from its parent. At an interactive shell, more than one of these can reach the same terminal.

Path	Conventional use	Change it in Shell+
0, standard input	Read a key or the next bytes of an input stream	**<** followed by an input filename
1, standard output	Deliver the command’s regular result	**>** followed by a new output filename
2, standard error	Deliver diagnostic messages, status and prompts	**>>** followed by a new diagnostic filename

Shell+ arranges the destinations before starting the child. A program that writes to path 1 naturally follows the new destination. A program that opens **/Term** explicitly has made a different choice. In particular, **>>** on this shell selects path 2. Treating it as a request to append to an existing report can produce an error or put the wrong stream in a file.

Filter a listing, then split a stream

From **/H1/station** run these commands. Choose an unused name for **archive** if repeating the exercise.

```
list report ! grep beacon
echo radio ! tee archive
list archive
```

The exclamation mark joins one command’s output to another command’s input. In the first pipeline, List sends both lines of the procedure down a pipe. Grep emits the line containing **echo beacon online**. The **pwd** line fails the match. In the second pipeline, Tee reads the line produced by Echo, writes it to the file **archive**, and also sends it to its own standard output. You see **radio** twice, once during the pipeline and once when List reads the saved file.



Figure 2: The running guest filters the report procedure, displays radio from a pipeline and reads radio back from the file named archive.

A pipe is a stream between open paths. OS-9 can schedule the reader while the writer is producing more bytes. Once every writer closes its path and the reader consumes the remaining bytes, the reader reaches end of input. A disk file has a directory entry and remains available to a future process. You can come back to **archive** after a reboot.

Catch a diagnostic separately

Use **build fail** in your practice directory. Give it one line, **list absent**, and then an empty Return. Leave **absent** as a deliberately unused filename. At Shell+ type:

```
shell <fail >>errors
list errors
```

The child shell tries to list a missing file. Most of the usual terminal chatter disappears. Opening **errors** afterward reveals its prompts, **ERROR #216**, and **eof**. In our session a short Shell+ banner still appeared on the terminal. This tells you why a useful diagnostic capture includes path 2 even when some messages still reach path 1. The file **errors** needs a fresh name for a second run.

The earlier filter, file copy and diagnostic capture all pass text through the same three numbered paths. Remember path 1. The report writer you assemble at the end of the book uses that number directly.

A file has a name and an owner

Chapter 4 of 14

Before changing a procedure, you want a known good copy. A name helps you find it. To decide whether you can read, change or run it, OS-9 consults information stored in its file descriptor.

Compare a working file and a copy

Begin in **/H1/station** with the original **report**. Use an unused destination name.

```
copy report backup
cmp report backup
attr backup
dir -e /h1/station
```

Cmp compared **00000017** bytes and found **00000000** differences. Those counts are hexadecimal. The two files each hold 23 bytes. A comparison of contents answers a narrower question than a directory listing. It says the bytes match. The detailed listing shows the files have distinct descriptor sectors and gives their owner, modification timestamp, attributes and size.

The attributes of **backup** read **------wr**. Read the positions from right to left: owner read, write and execute, followed by public read, write and execute, the share-mode bit, and the directory flag. Here only the rightmost two bits are set. When OS-9 checks a file, it considers the user ID of the caller as well as the access requested. A permission to write an existing file and a permission to create entries in a directory answer different questions.

Give the backup a durable name

```
rename backup report.old
shell report.old
echo new >report
list report
```

Rename changes the directory entry. The saved copy still runs and prints **beacon online** and your working directory. The attempted create over **report** returns error 218. The original procedure still lists both commands. On this shell a fresh filename is the simple way to keep a second run’s output.

```
 Directory of /h1/station  2026/11/06 19:27

Owner  Last modified   Attributes Sector Bytecount Name
----- ---------------- ---------- ------ --------- ----
   0  2026/11/06 19:27  ------wr      28        17 report
   0  2026/11/06 19:27  ----r-wr      30         7 archive
   0  2026/11/06 19:27  ------wr      38        17 backup


(Term!02)/H1/station:rename backup report.old

(Term!02)/H1/station:shell report.old
beacon online
/H1/station

(Term!02)/H1/station:echo new >report
ERROR #218

(Term!02)/H1/station:list report
echo beacon online
pwd

(Term!02)/H1/station:
```

Figure 3: Shell+ shows the renamed procedure running, refuses to create over report, and lists both original lines afterward.

Diagnose a permission failure

Use a separate copy so you can observe a real access error and repair it without changing either copy of the lesson procedure. The following assumes **guard** is unused.

```
copy report guard
attr guard -r
list guard
attr guard r
list guard
```

Removing the owner read bit produced **------w-**. The first List returned **ERROR #214**. Restoring the bit produced **------wr**. The last List recovered the original two lines. The name and bytes survived throughout. This is a different problem from error 216, where the requested pathname could not be found.

For a program module, owner execute is a separate attribute. Chapter 13 sets that bit on the station's own assembled program. The right to execute a file and the operating system's ability to recognize its module header are both relevant there.

Inspect a supported image without booting it

For a supported raw OS-9 image, **Edit...** beside its mounted floppy or hard drive provides a native directory browser. For a separate unmounted image, use **Media → Floppy Disks... → Open Disk Image...**. If the image contains more than one filesystem, select its OS-9 volume before browsing the folders. Select a file or folder and choose **Info**. The same File Information sheet shows its details, owner/public read-write-execute bits, **Single access (non-shareable)** and ownership. File versus Directory is structural information and stays read-only. Single access permits only one opener at a time.

OS-9 stores one 16-bit owner ID. The editor's Group ID and User ID fields are its high and low bytes, each from 0 through 255; the table displays **group.user**. Group 12 and user 34 therefore mean decimal owner ID 3106. This is not a Unix owner/group/other permission layout: access has owner and public triplets.

Save Changes in File Information changes the editor's private working image. Undo/Redo can reverse it before the image itself is saved or applied. For a mounted floppy or hard disk, the editor pauses and locks the CoCo and requires an explicit restart before applying, so OS-9 cannot reuse an old descriptor or allocation cache. Image write protection still applies even when the host editor can repair an entry with no guest access permissions.

For regular files within that supported image, **Move To…** in the editor's toolbar or context menu chooses another OS-9 folder. Dragging selected files onto a folder uses the same operation. Permissions and ownership remain attached to the file; Undo restores the previous locations. A duplicate destination name is refused without overwriting, and the move remains private until the image is saved or applied. See Move files between OS-9 folders.

The editor identifies filesystems within supported raw hard-drive images, including the OS-9 region and numbered HDB-DOS Disk BASIC volumes when both share one image. Space totals describe the selected filesystem; saving an edited copy keeps the complete image and preserves its other regions. It does not format, repair or convert a disk. Follow the Disk Image Editor procedure and limits. The supplied station exercises continue to use guest commands, so you can observe OS-9's own file handling and permissions.

Follow a filename onto the disk

Chapter 5 of 14

What did OS-9 actually do when you asked for **/h1/station/report**? The text occupies 23 bytes. To reach those bytes, the operating system first had to read two directories and a file descriptor. You can reconstruct its route from a real working disk without writing a single sector.

Compare the file and the volume

In the station directory, enter:

```
free /h1
dir –e /h1/station
dump report
```

Our fresh practice filesystem identified itself as **VCC Shared Data**. Its capacity was **368,640 sectors** in clusters of eight. A sector here contains 256 bytes. A cluster therefore accounts for 2,048 bytes of allocation. Your free-sector count will change as files and directories appear. These figures describe the OS-9 filesystem inside the shared image. Other parts of that same image can hold BASIC volumes.

Dump starts counting at byte zero of the *file*. Look for **65 63 68 6F**, the ASCII bytes for **echo**. Look for **0D** after **online** and again after **pwd**. Each carriage return is one byte. The length shown as hex **17** in the directory agrees with 23 bytes, including those two line endings.

```
(Term!02)/H1/station:dir -e /h1/station

 Directory of /h1/station  2026/11/06 19:30

Owner  Last modified   Attributes Sector Bytecount Name
----- ---------------- ---------- ------ --------- ----
   0  2026/11/06 19:27  ------wr      28        17 report
   0  2026/11/06 19:27  ----r-wr      30         7 archive
   0  2026/11/06 19:27  ------wr      38        17 report.old


(Term!02)/H1/station:dump report

Address   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0 2 4 6 8 A C E
--------- ---- ---- ---- ---- ---- ---- ---- ----  ---------------
00000000 6563 686F 2062 6561 636F 6E20 6F6E 6C69  echo beacon onli
00000010 6E65 0D70 7764 0D                        ne.pwd.

(Term!02)/H1/station:
```

Figure 4: The guest's extended directory and hex dump show a 17-hex-byte report whose last bytes are 70 77 64 0D, the word pwd and carriage return.

Walk from a device name to the first text byte

The table was decoded from the actual HD1 image used to make that screen. Numbers beginning with **$** are hexadecimal. The operating system assigns sectors as the disk changes, so use the chain of references when reading a different image. An extra file created earlier can move every later allocation.

Read this object	Find this field or entry	Follow it here on the example image
Volume description, logical sector 0	Three bytes at offset $08 name the root directory's descriptor	**00 00 18** → sector $18
Root directory descriptor at $18	Its first segment starts at $19 and spans seven sectors	Read root directory records beginning at $19
Root directory record named **station**	Last three bytes name its descriptor	**00 00 20** → sector $20
Station directory descriptor at $20	First segment starts at $21	Read 32-byte directory records beginning there
Station directory record named **report**	Last three bytes identify its descriptor	**00 00 28** → sector $28, also shown by the guest's detailed listing
Report descriptor at $28	File length is $17. First segment starts at $29 and spans seven sectors	Read only the first 23 bytes at $29
File data at $29	The bytes start with **65 63 68 6F** and end in **70 77 64 0D**	These are the same bytes printed by the guest's Dump

For a raw image with this filesystem starting at byte zero, sector $29 begins at byte **$2900**. The extent describes allocated sectors. The file-size field tells a reader where valid file content ends. Space reserved after a short file's last byte is not part of that file.

Inspect the sector on your CoCo

On a personal practice image, use the disk editor as a viewer. Enter **ded /h1@** at Shell+. The trailing **@** ** selects the entire device. The first screen identifies logical sector zero in the upper left. Use row and column labels to read the three bytes starting at $08. To follow their value, press** S**, enter the hexadecimal sector number at the sector prompt, and press Return. For example, our first jump was** 18**. Use** S** again to follow each link in the table, taking the values from your own disk. Directory records are 32 bytes each. If a name is farther down the directory, continue in its data segments.

At the sector containing the first bytes of **report**, compare the right-hand character display and the first 23 hex bytes with the file dump you saw earlier. The disk view can display 256 bytes even though this tiny file ends much earlier. Press **Q**, then **Y** at the quit confirmation to return to Shell+. These steps only read sectors. The editor's other modes can write raw disk structures. Use the navigation keys above for this exercise.

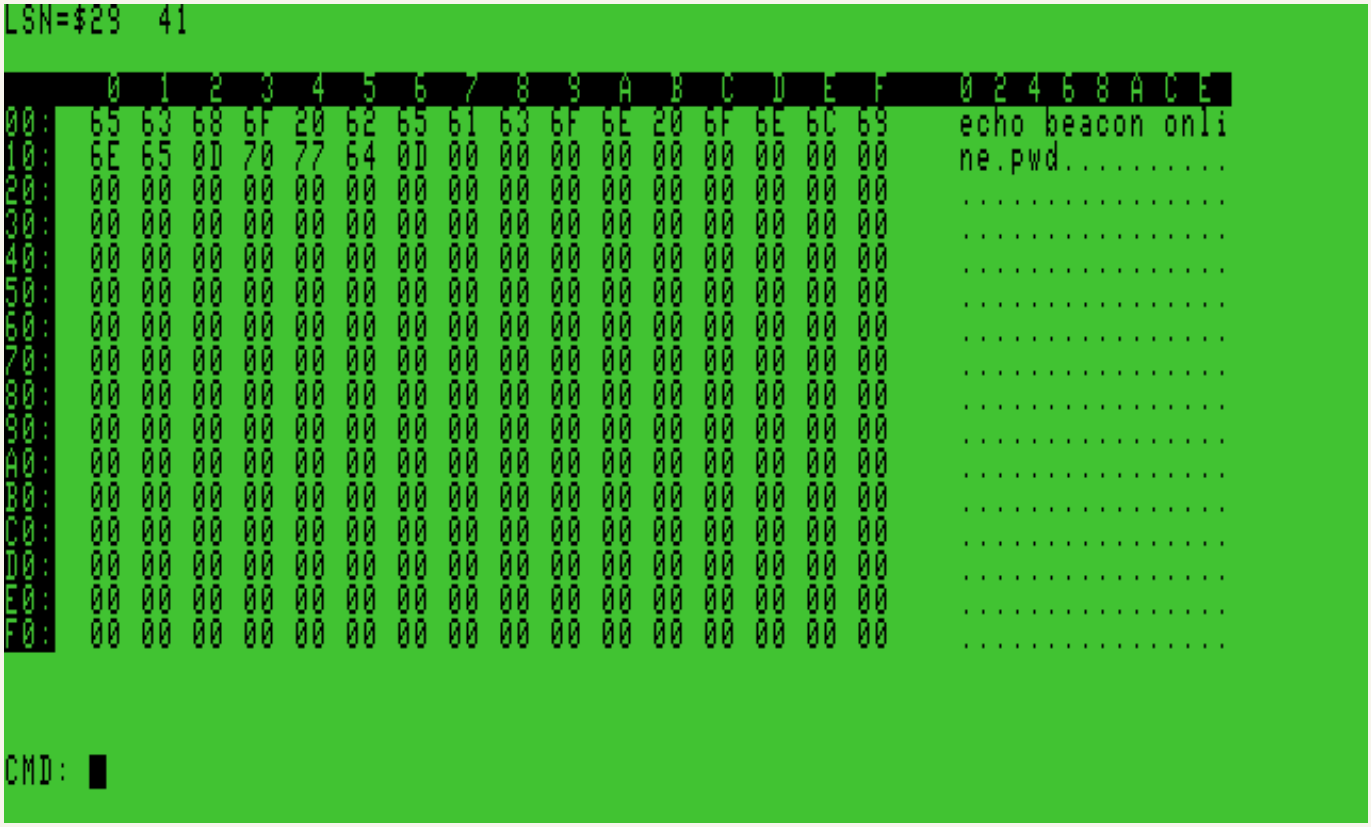


Figure 5: dEd shows sector $29, decimal 41. Its first two rows contain the station procedure and the rest of the displayed sector is zero in this copy.

Where the allocation map fits

Sector zero records the size of the map in bytes (**$1680**) and the number of sectors represented by one bit (**$0008**). $1680 bytes times eight bits per byte times eight sectors per bit accounts for the reported 368,640 sectors. Here the map starts at sector 1. A set bit marks an allocated cluster. The first bitmap byte in the pictured exercise was **FF**. It covered sectors $00 through $3F, including the volume structures and the practice files. When opening an existing file, RBF follows directory entries and segments. When it needs more space, the bitmap helps it locate free allocation units. A directory entry does not record a run of text sectors, and the bitmap does not say which filename owns a particular cluster. Together these structures answer different questions about the disk. The storage atlas gives the field offsets for the example.

Recover the right copy

Chapter 6 of 14

Suppose the relief operator remembers a report from the first shift, but the current disk also contains second-shift files. Recover the earlier state while keeping the later one. Give each state an unmistakable filename on the Mac and an observable difference inside OS-9.

Take a whole-image backup

Use your **personal practice** NitrOS-9 profile. At Shell+, finish the station commands from the previous chapters. Open **Media → Virtual Hard Disks…**, record the filename and location assigned to **HD1** and look for **Saves automatically**. The profile's **Files** list gives the path if the drive row only shows a filename. Wait for the save to finish. If you chose **Temporary changes**, use **Save As…** to retain those bytes and verify the actual HD1 assignment before continuing.

Quit the emulator. In Finder, duplicate the identified **HD1** image and name the duplicate **Station First Shift.vhd**. Keep it as the backup. The original remains at the pathname the practice profile knows. Copying a disk after the guest has stopped gives you a consistent point in time. The copy includes filesystem structures and the BASIC volumes that share this disk image.

Give the later state a recognizable marker

Reopen VCC Plus, load the same practice profile, run it and wait for Shell+. Type:

```
chd /h1/station
echo second shift >shift.two
list /h1/station/shift.two
```

The listing reads **second shift**. Let the automatic save finish and quit the emulator again. In Finder, rename the current, later HD1 image **Station Second Shift.vhd**. Keep it too. Duplicate **Station First Shift.vhd** and give that duplicate the *exact original HD1 filename* in the *original folder*. The profile can now open a writable restoration of your backup while both named snapshots remain intact. A different directory containing an identically named image will not satisfy the saved path.

Verify from a fresh boot

Open VCC Plus again, load the same practice profile, and choose **Run**. Verify the HD1 assignment. After OS-9 boots enter:

```
dir /h1/station
list /h1/station/shift.two
list /h1/station/report
```

The directory on the restored image contains **report**. Asking for **shift.two** gives **ERROR #216**, and the final listing shows **echo beacon online** and **pwd**. The missing marker is the desired result here. You just identified a generation of the disk by its contents. The later snapshot still holds the marker.

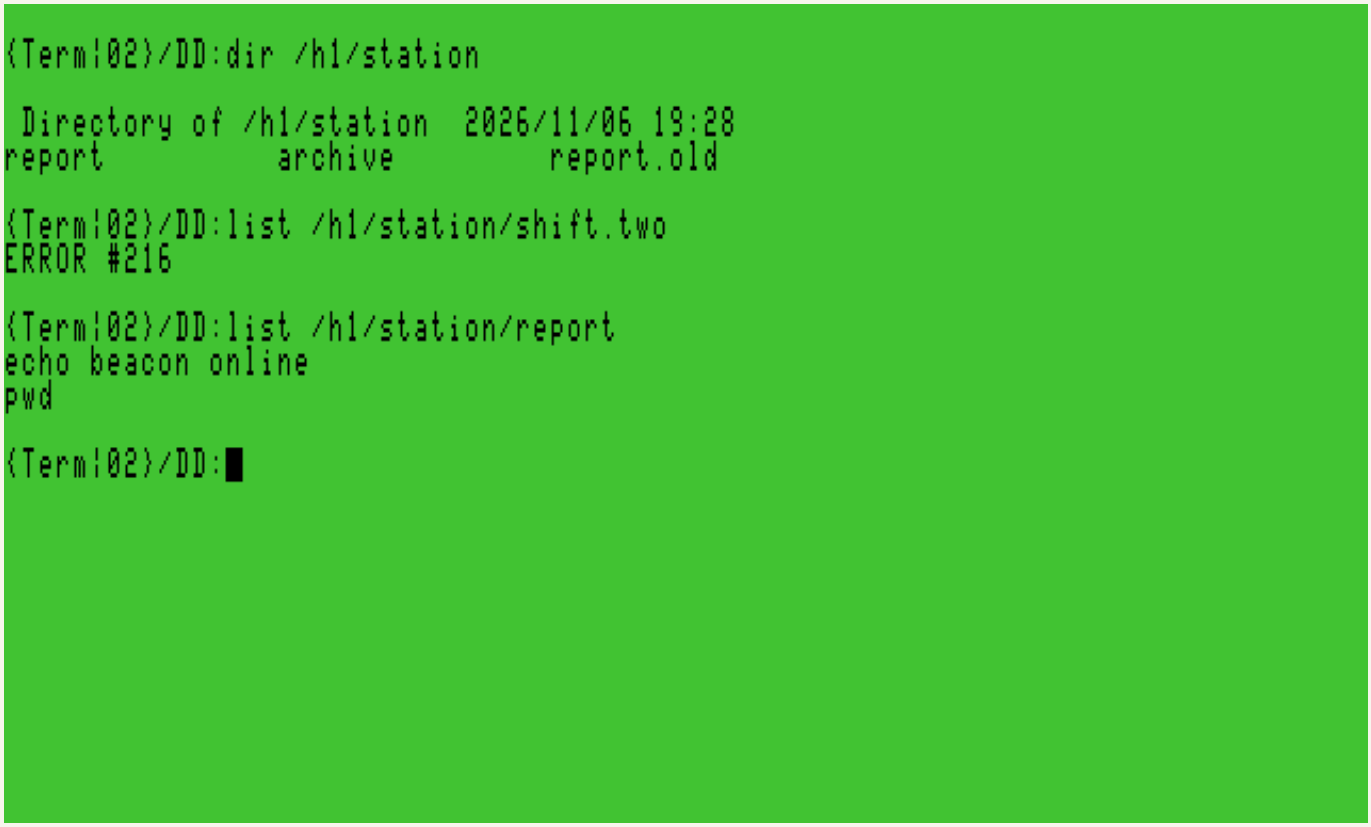


Figure 6: After booting the restored HD1, Shell+ lists the station directory, reports error 216 for the newer marker, and retrieves both original lines from report.

If the marker still exists, inspect the HD1 pathname and confirm which copy you duplicated. **If both paths are missing**, check whether you loaded fresh starter media or mounted another personal image. **If writes are pending**, finish or export them before replacing an image. A profile records the hardware and media assignments. The disk backup supplies the actual directories, permissions and file data for that point in time. This recovery leaves the shell working in **/DD** after boot. Use **chd /h1/station** before resuming exercises that name station files without their directory. You can keep the restored copy as the active practice disk for the remaining chapters.

Catch a process while it sleeps

Chapter 7 of 14

You can leave a job running on the CoCo and still type another command. Try this with a program whose only job is to wait. You will have time to find it in the process table, see who started it, and finish it deliberately.

What came back?

Begin at Shell+ in your practice session. Enter each line separately.

```
sleep 1
echo %*
list /h1/absent
echo %*
```

Use a practice disk where **absent** is an unused filename. After the short sleep, the first **echo** prints **000**. After **list** reports **ERROR #216**, the second prints **216**. **%*** expands to the last command's status in this Shell+. Inspect it immediately. Running another command can replace the evidence.

OS-9 gives each running instance of a program a **process ID**. When the shell starts an external program in the foreground, the child inherits paths such as standard output. The parent waits for the child to finish and can collect its exit status. A command that prints nothing can succeed, as Sleep just did.

Let two jobs overlap

```
sleep 3600 &
procs
```

The ampersand lets the parent shell accept another command while the child runs. The shell reports a number such as **&003**. Look in the **Primary Module** column for **Sleep**. Read across to **Id**, its own ID, and **PId**, its parent. During the illustrated run, Sleep was 3 and its parent was 2. Procs itself was another child of 2. It had to be running to report the table.

```
&005
&006
&007

Shell+ v2.2a 26/11/06 19:30:22

(Term!02)/DD:sleep 1

(Term!02)/DD:sleep 3600 &
&003

(Term!02)/DD:procs

          User                        Mem Stack
Id  PId Number  Pty Age Sts Signl Siz  Ptr   Primary Module
--- --- ------- --- --- --- ----- --- ----- ---------------
  2   1       0   128 128 $80     0   31 $63E2 Shell
  3   2       0   128 128 $C0     0   31 $60DF Sleep
  4   2       0   128 128 $80     0   31 $1EF3 Procs
  5   0       0   128 130 $80     0   31 $5CE2 Shell
  6   0       0   128 130 $80     0   31 $58E2 Shell
  7   0       0   128 129 $80     0   31 $53E2 Shell

(Term!02)/DD:
```

Figure 7: The guest process table shows Shell as ID 2, Sleep as ID 3 with parent 2, and the inspecting Procs command as ID 4 with the same parent.

The **Pty** and **Age** fields belong to the scheduler. A runnable process competes for processor time. A sleeping or waiting process can give up its turn until its event occurs. Giving two processes the same base priority does not make their progress identical. The hour-long sleeper spends its time waiting. The interactive shell can be waiting for a key at the same moment.

End only the child you started

Type **kill**, a space and the **Id you just observed for Sleep**. Press Return and run **procs** again. The Sleep row should disappear. Other Shell rows belong to the prepared system's other sessions. Leave those running.

The shell can report a child termination and error **228** as it collects the killed child. In this exercise that is the expected consequence of asking a process to stop. A signal and an exit status have different jobs. One delivers an event to a process that is still alive. The other tells a waiting parent the outcome after the child finishes.

At the operating-system interface, **F$Fork** creates a process, **F$Wait** lets the parent collect a child's termination, and **F$Exit** ends the calling process with its status in B. You will write that last call in the beacon report program. For the register contracts see The NitrOS-9 Project, *NitrOS-9 Technical Reference*, revision 0.1 (2004), pp. 99–100 and 128.

Where does a program live?

Chapter 8 of 14

A copy of Echo is on disk. There is also an Echo in the running system's module directory. Its instructions can serve one shell and then another. Each invocation still needs its own arguments, writable storage and stack.

Read a module's identity

```
ident /dd/cmds/echo
mdir
```

ident examines the named disk file. In the supplied edition, look for the name **Echo**, size **$0022** (34 bytes), execution offset **$0012** and a good module CRC. Offset $0012 means the first instruction is 18 bytes from the start of the module. The header tells the loader where to begin. The CRC checks the module's bytes against the check value recorded in the file.

mdir lists modules currently known to the kernel. The prepared boot loads a group of frequently used utilities, so Echo is already there. A file may contain several modules, each with its own header, name and entry in memory. A module can also be a file manager, driver or device descriptor. A name present in **mdir** alone tells you little about whether that name makes sense at a command prompt.

Count references you create yourself

Use a fresh practice boot with no other job using **Cmp**. Here Cmp is the small file-comparison program. Use the lowercase letters **mp** in the filter to match the ending of **Cmp** as it is printed.

```
load cmp
mdir -e ! grep mp
link cmp
mdir -e ! grep mp
unlink cmp
mdir -e ! grep mp
unlink cmp
mdir -e ! grep mp
```

Read the last number before the name. The four checks on our guest gave **1**, **2**, **1**, and finally no Cmp row.

```
Data Size:   $01C2    #450
Edition:     $05      #5
Ty/La At/Rv: $11 $81
Prog mod, 6809 obj, re-en, R/O

(Term!02)/DD:load cmp

(Term!02)/DD:mdir -e ! grep mp
:   9    0  27C 11  0 r   1 Cmp

(Term!02)/DD:link cmp

(Term!02)/DD:mdir -e ! grep mp
:   9    0  27C 11  0 r   2 Cmp

(Term!02)/DD:unlink cmp

(Term!02)/DD:mdir -e ! grep mp
:   9    0  27C 11  0 r   1 Cmp

(Term!02)/DD:unlink cmp

(Term!02)/DD:mdir -e ! grep mp
```

Figure 8: Four queries for Cmp show its use count climb from 1 to 2 and drop back to 1. The last query has no matching row.

load brought the program from the execution directory into memory and retained a reference. **link** found that named module in memory and retained a second reference. Each **unlink** released one. The disk file remains available after the module leaves memory. If other processes have acquired the module, their references also count. Release only the references you created. For boot modules, keep the system's existing references intact.

Give each caller somewhere to write

A re-entrant program can serve multiple callers using one copy of its code. Position-relative addressing lets a reference inside that code keep pointing to the same item after the module is placed in memory. Per-process data and a private stack give different executions a place to store different answers. Code that stores a changing counter directly into its shared instructions or embedded constants would break that arrangement.

The directory of running **processes** therefore answers a different question from the directory of **modules**. In the preceding chapter you found several Shell processes. In the next chapter you will see several of them map the same physical code block. For the header layout and module CRC see The NitrOS-9 Project, *NitrOS-9 Technical Reference*, revision 0.1 (2004), chapter 3, pp. 26–32.

One address, two views

Chapter 9 of 14

Two Shell processes can have different private data while executing the very same instruction bytes in RAM. The CoCo 3 makes that possible with its memory management unit. NitrOS-9 Level Two keeps the appropriate block assignments for each process and installs the mapping it needs when switching execution.

Read a process map

```
pmap
mfree
```

Read **pmap** across. Each pair in the heading names the leading hexadecimal digits of one 8 KiB logical range. **01** covers $0000–$1FFF. **23** covers $2000–$3FFF. The last column, **EF**, covers $E000–$FFFF. The numbers in each row are physical block numbers. Dots mark positions without a block assigned to that process by OS-9. In the example, Shell processes 2 and 5 have different blocks at the left, **07** and **0D**, and the same block **0A** at the right. They can each have a private stack or data while sharing their code. **mfree** instead counts available physical memory in system-wide blocks. A 512 KiB CoCo has 64 blocks of 8 KiB. Installing another block somewhere in that machine does not give one 16-bit process a ninth slot in its logical map.

Predict an instruction’s physical address

Use the supplied system and the two-line station procedure you made earlier. In the station directory, inspect Shell's placement and its entry offset.

```
mdir -e ! grep hell
ident /dd/cmds/shell ! grep 0074
```



Figure 9: PMap shows multiple Shell processes with top block 0A. The module directory places Shell at offset zero in that block, and Ident reports an entry offset of $0074.

Here Shell starts at offset zero in physical block **$0A**. The **Shell** header in the supplied file has execution offset **$0074**. Since that block occupies the top slot in the process, its entry is **$E000 + $0074 = $E074**. If the image or Shell edition differs, inspect the header and map for that edition before setting a breakpoint. A file with multiple modules can produce several sets of header information.

Quantity in this run	Value
Start of physical block $0A	$0A × $2000 = $14000
Offset into the physical block	$0074
Predicted physical instruction address	**$14074**

Stop while that map is active

1. Open the machine **Debugger**. In **Breakpoints**, add the logical address **E074**. Resume if you paused to enter it.
2. At the CoCo shell, run **shell report** in the station directory. The child shell reaches its entry instruction and the emulator stops there.
3. Read **PC** and the **MMU** inspector. Our run stopped at **E074** with **task 1** active. The effective bank for logical $E000 was **0A**. A stop inside an interrupt handler or a different process can show a different map.
4. In **Memory**, inspect **Mapped CPU** at **E074**, then **Physical RAM** at **14074**. Both views began **32 7B 34 36 30 C9 18 15**. Checking eight bytes at a predicted address gives you a better fingerprint than one matching byte.
5. Remove the breakpoint and **Continue**. The procedure prints **beacon online** and its working directory, and the parent prompt returns.

A debugger breakpoint is specified in logical CPU space. Context matters if another process also executes at the same address. The GIME's two hardware task selections do not limit the OS to two processes. OS-9 can reload registers from saved mappings as processes take turns. When a PMap position is unused, a hardware register may still show a value programmed by the OS. Use the process map to interpret ownership and the debugger's effective mapping to interpret a particular CPU read.

Book 10 shows the same translation with a byte deliberately written by a BASIC-hosted program. For the CoCo 3 MMU under Level Two see The NitrOS-9 Project, *NitrOS-9 Technical Reference*, revision 0.1 (2004), chapter 2, p. 13.

Open a device by name

Chapter 10 of 14

Imagine removing the output cable from a terminal and plugging it into a disk. Programs would need radically different electrical signals for those two devices. OS-9 gives a program a simpler operation. Write bytes to an open path. The system finds the right file manager and driver for that path. See the dispatch in the live device table.

Catch the drive while it is open

Use a fresh filename on HD1. From Shell+ enter:

```
devs >/h1/devices
list /h1/devices
dmode /h1
```

There is a reason to save this particular report. While **devs** examines the active device table, its output file holds H1 open. If you just run **devs** at a prompt, an idle H1 may be absent from the display. The saved report records the conditions at the instant the utility ran.

```
(Term!02)/DD:mdir -e ! grep mp
:    9     0  27C 11   0 r    1 Cmp

(Term!02)/DD:unlink cmp

(Term!02)/DD:mdir -e ! grep mp

(Term!02)/DD:devs >/h1/devices

(Term!02)/DD:list /h1/devices

Device Table at: 6B00

Dv Desc  Drvr    Stat  File  Usr
Nm Port  Name     Mem  Mgr   Cnt
-------  -------  ----  ---
DD FFE0  EmuDsk 6900  RBF    06
Te FFA0  VTIO   6800  SCF    04
W2 FFA2  VTIO   5A00  SCF    03
W3 FFA3  VTIO   5900  SCF    03
W4 FFA4  VTIO   5000  SCF    02
H1 FFE0  EmuDsk 6900  RBF    01

(Term!02)/DD:
```

Figure 10: The guest's saved device table pairs DD and H1 with EmuDsk and RBF. Term and the window devices use VTIO and SCF.

Look for **H1**, **EmuDsk** and **RBF** on one row. The descriptor names the device and specifies its driver, file manager and operating parameters. **RBF**, the random block file manager, understands directories, files and disk sectors. **EmuDsk** is the driver that talks to the emulated virtual disk controller. In this profile, **dmode /h1** reports drive number **01**, and VCC Plus assigns the second working image to HD1. A different controller can use a different driver under an RBF filesystem.

Trace three paths from the same shell

If the program sends bytes to…	Follow the request
A file under **/h1**	IOMan dispatches through RBF. RBF resolves the file and its sectors. EmuDsk transfers blocks through the virtual hard-disk device to the image assigned to HD1.
The terminal **/Term**	SCF handles a stream of characters and its terminal options. The VTIO driver and the CoCo keyboard and display components deliver the interaction. Read the Term row next to H1 in the figure.
A shell pipeline	PipeMan manages a buffered stream with the supplied Piper driver. The writer sends bytes to one end. The reader takes them from the other. Closing the last writer lets the reader detect the end of the stream.

The program refers to an open **path number** during I/O. It need not resolve the pathname or understand sectors each time it writes a line. The shell can arrange path 1 before launching the child, which is how the same program can write its report to any of those three destinations. A **device descriptor** is a named module that tells OS-9 how to reach a device. An **open path** holds the state of a particular opening, such as a file's current position and access mode.

The graphic panels in VCC Plus show which host images are attached to the emulated drives. The guest device table shows the software OS-9 is using to reach them. Compare the quick-start equipment map with the same two assignments in your current profile. For the division of responsibilities see The NitrOS-9 Project, *NitrOS-9 Technical Reference*, revision 0.1 (2004), chapters 4–7, pp. 33–86.

Who owns this terminal?

Chapter 11 of 14

OS-9 can run several shells on one CoCo 3. In the prepared boot, Shell+ starts on **/Term** and additional shells run on window devices. All of them share one physical keyboard. OS-9 chooses which guest window receives it. macOS separately decides whether your typing reaches the emulator at all.

Identify your terminal

Click the CoCo display and enter **tmode**. The first line on the main console is **/Term**. The options below include **eko** for character echo, **eor** for the end-of-record character, and **eof** for the end-of-file character. Here **eor=0D** is a carriage return. These are options on the current terminal path. You can inspect them without changing them. Run **devs** to inspect the active window devices. On the supplied profile, the startup procedure uses W2, W3 and W4. Use the practice system with **W7** free for this exercise. If W7 is already in use by your work, choose a fresh practice session. A window may display output even when it has no shell reading the keyboard.

Draw the station’s signal meter

At the main terminal enter the following two commands. **iniz** holds an attachment to the new device. **wcreate** defines its screen and colors. Screen type **8** selects a 320 × 192, 16-color graphics screen. The rectangle in the creation command is measured in character cells. It starts at the top left and spans 40 columns by 24 rows. The final three values select the foreground, background and border palette entries.

```
iniz /w7
wcreate /w7 -s=8 0 0 40 24 03 00 00
```

Send three pairs of drawing commands to W7 while you are still typing in /Term. The numbers after **display** are hexadecimal bytes. **1B 40** sets the draw pointer, and **1B 4A** fills a rectangle from that pointer to the supplied opposite corner. Each coordinate consists of a high byte and a low byte. The driver interprets positions under its current scaling. The three bars share the same lower edge at Y = $0088.

```
display 1b 40 00 60 00 68 1b 4a 00 a0 00 88 >/w7
display 1b 40 00 c0 00 48 1b 4a 01 00 00 88 >/w7
display 1b 40 01 20 00 28 1b 4a 01 60 00 88 >/w7
display 1b 21 >/w7
```

The last command selects W7 for display. Look for three bars rising from left to right. In the captured palette the bars are green and the background is white. The staircase is the useful test even if your colors differ.



Figure 11: The actual guest graphics window displays three solid green bars. Their lower edges align and each successive bar is taller.

Return to the shell and release the window

With the guest display focused, press **CoCo Clear**. VCC Plus maps the Mac **Home** key to Clear. On the prepared profile we returned to the original terminal in one press. If your active windows are ordered differently, cycle until you see the original **Term** prompt. Trying to type a cleanup command on the graphic would send keys to the wrong window.

Back on /Term enter:

```
display 1b 24 >/w7
deiniz /w7
tmode
```

1B 24 ends the display window. **deiniz** releases the attachment you took with **iniz**. **tmode** should again identify **/Term**. You have created a graphics device, drawn into it, selected it, regained keyboard control and released the device while the same guest operating system continued running.

A second OS-9 window still belongs to the CoCo. The optional guest connection to the Mac shell crosses into macOS and has its own permission control. Use the terminal name and the command interpreter to decide which machine will receive your next command.

For the window protocol, see The NitrOS-9 Project, *The NitrOS-9 Level 2 Windowing System*, pp. 7–8 (creation), 20–23 (end and setup), 44 (selection), 49 and 58 (bar and draw pointer).

Put the night watch on duty

Chapter 12 of 14

Your station needs someone to count three watches. In BASIC09 you can give the job a name, declare the storage it needs, and call it from either the language workspace or the shell. The same three lines of output will tell you when each version works.

Write the procedure on HD1

Start at Shell+ with the writable practice disk in HD1. Use new names for **night.b09** and **night** so you can keep both the original and the packed copy. Enter one command per Return.

```
chd /h1
build night.b09
```

BUILD prompts with a question mark for each new line. Enter these six lines, then press Return at the empty prompt. The file is also available as night.b09.

```
procedure night
dim watch:integer
for watch=1 to 3
print "watch ";watch
next watch
end
```

The first line gives the procedure the name **night**. Its integer variable **watch** belongs to this procedure. The **for** assigns 1 before the first trip through the loop. Each **next watch** advances the counter and tests whether another iteration is due. Each trip prints a word and the current value on one line. **end** finishes the procedure. The loop has a name and a matching end point on the page, so you can follow it without line numbers.

Find it in the BASIC09 workspace

At Shell+ make the practice disk your execution directory and launch the language system from the system disk.

```
chx /h1
/dd/cmds/basic09
```

There are two independent locations here. The data directory is where BASIC09 opens **night.b09**. The execution directory determines where the packed version will be written. The absolute name **/dd/cmds/basic09** still locates the language system after you change directories.

Look for **Ready** and the **B:** prompt. Enter:

```
load night.b09
run night
```

LOAD reads the text file and recognizes the procedure by the name inside it. RUN takes that procedure name. You should see:

```
watch 1
watch 2
watch 3
```

Ready and **B:** appear again. You are still in BASIC09. Compare the way you entered the source with programming in CoCo BASIC. On this disk BASIC09 is an OS-9 program with its own workspace and commands. The CoCo BASIC editor in the Mac Develop menu works with the ROM BASIC interpreter.

Hand it to the runtime

At **B:** enter the next two commands.

```
pack night
bye
```

PACK writes a module containing the procedure's intermediate code. BYE leaves the language workspace. Now launch the smaller BASIC09 runtime from Shell+.

```
/dd/cmds/runb night
```

This time you should see three watches and then the **OS-9 shell prompt**. RUNB looked up the packed procedure by name and ran it without opening the editing workspace. In this configuration giving RUNB the full path as its procedure argument produced error 043. Use the execution directory and the procedure name shown above.



Figure 12: The green guest display shows three watch numbers from BASIC09, the pack and bye commands, and the same three numbers from RUNB at the shell.

Restore normal command lookup when you finish.

```
chx /dd/cmds
dir /h1
```

Find **night.b09** and **night**. Keep the first file to edit the original text. The second can run under RUNB. In the next chapter you will give the station an executable module whose body consists of 6809 instructions.

For the language and its workspace, see Microware Systems Corporation, *BASIC09 Programming Language Reference Manual*, revision H (1984), “Introduction to BASIC09 Programming” and “Program Statements and Structure.”

Assemble a program on the CoCo

Chapter 13 of 14

Write a station report in 56 bytes. Most of those bytes will describe the program or hold its message. Only seven instructions are needed to deliver the line and return control to the shell.

Give OS-9 a module

Begin at Shell+ with **/dd/cmds** as the execution directory and a personal HD1. Keep any previous **beacon.a** or **beacon** by using a fresh practice image for a repeat. At the shell enter:

```
build /h1/beacon.a
```

Type the Beacon source one line at a time. Start each label, such as **start** or **finish**, in the first column. Keep a leading space on lines that start with an assembler directive or an instruction. Finish by pressing Return at the next empty BUILD prompt.

```
        mod endmod,name,$11,$81,start,256
name    fcs /beacon/
        fcb 1
start   lda #1
        leax note,pcr
        ldy #notesiz
        os9 $8c
        bcs finish
        clrb
finish  os9 $06
note    fcc /beacon report/
        fcb 13
notesiz equ *-note
        emod
endmod  equ *
        end
```

mod starts a module. Its arguments supply the end of the module, the name, its type and language, its attributes and revision, the entry point, and a request for working storage. The assembler resolves the labels as offsets. **fcs** stores the module name with the high bit of its last character set. The byte immediately after the name gives edition 1. Those bytes belong to the name and header information. The first executable instruction begins at **start**.

The header declares a 6809 program module ($11), re-entrant with revision 1 ($81). OS-9 can share its program bytes among callers. Each process gets its own working storage and stack. Beacon requests 256 bytes and leaves its message inside the read-only module. **fcb 13** adds the carriage return expected by the line-writing service. The assembler evaluates **notesiz** as 14, the length including that final byte.

Follow the two system calls

At this line	What the register or flag means
lda #1	A selects path 1, the process's standard output. The shell chose what is on the other end.
leax note,pcr	X becomes the address of the first message byte. The displacement is relative to the program counter, so the module can execute at different logical addresses.

At this line	What the register or flag means
ldy #notesiz	Y supplies the maximum number of bytes the write may consume.
os9 $8c	I$WritLn asks the operating system to write a line. The assembler emits a 6809 SWI2 instruction followed by the service number.
bcs finish	The service sets carry on failure and puts the error number in B. This branch preserves the number for the caller.
clrb	On the success path the exit status becomes zero.
os9 $06	F$Exit ends the process and passes the status in B to its parent.

The CPU never has to know an address for the screen. In the BASIC ROM example, a caller used a ROM vector to send a character. Here the OS-9 process selects a path and asks the kernel for a service. In the next exercise the shell will put a disk file on that very same path.

Let the guest assembler check your work

```
list /h1/beacon.a
asm /h1/beacon.a o=/h1/beacon
ident /h1/beacon
```

Use the listing to catch a missing label or misplaced space before assembling. In the **asm** command **o=** names the output file. The final **emod** in the source generates the closing CRC. Here are four checks on what came out:

Field from IDENT	Expected value	How to read it
Header name and module size	**beacon**, **$0038**	The whole module, including header and CRC, occupies 56 bytes.
Execution offset	**$0014**	The first instruction is 20 bytes from the beginning of the module.
Data size	**$0100**	The process requests 256 bytes for working storage including its stack. These bytes are allocated at execution.
Module CRC	**$3D8059 (Good)**	IDENT recomputed the CRC successfully. This particular value identifies the bytes produced by the printed listing.

```
        emod
endmod  equ *
        end

(Term!02)/H1:asm /h1/beacon.a o=/h1/beacon
00000 error(s)
00000 warning(s)
$0038 00056 program bytes generated
$0000 00000 data bytes allocated
$00C2 00194 bytes used for symbols

(Term!02)/H1:ident /h1/beacon

Header for:  beacon
Module size: $0038    #56
Module CRC:  $3D8059 (Good)
Hdr parity:  $10
Exec. off:   $0014    #20
Data Size:   $0100    #256
Edition:     $01      #1
Ty/La At/Rv: $11 $81
Prog mod, 6809 obj, re-en, R/O

(Term!02)/H1:
```

Figure 13: The guest assembler reports zero errors and zero warnings. IDENT shows the 56-byte beacon module, entry offset $0014 and good CRC $3D8059.

The assembler's line counting allocated data can read zero even though the header requests $0100 bytes at runtime. Compare what was **written into the file** with what a **new process will need**. Now take your completed module to the station console.

For the format and service conventions, see The NitrOS-9 Project, *NitrOS-9 Technical Reference*, revision 0.1 (2004), chapter 3, pp. 26–32, F$Exit, p. 99, and I$WritLn, p. 148.

Deliver the beacon report

Chapter 14 of 14

You have a file that contains a program. The filesystem controls who may execute the file. Inside the file, the module header tells OS-9 what it can load. Those are two different checks. Give the file execute permission and run it from its full path.

Choose the destination

```
attr /h1/beacon e
/h1/beacon
```

The program prints **beacon report** and the shell prompt returns. Now run it again. This time let the shell create a fresh file named **report** at the root of your practice HD1.

```
/h1/beacon >/h1/report
list /h1/report
```

The first command goes straight back to the prompt. The second displays the two words. The file **/h1/report** contains the message from the module. Your shell procedure from the beginning of the book lives one directory deeper under **/h1/station/report**.

Trace one line through the system

Stage	On the terminal run	On the file run
Shell+ prepares the child	Standard output refers to the terminal.	The shell opens the named output file and supplies it as standard output.
The module begins	A new process starts at the entry offset in Beacon's header.	The same module, entry point and instructions are used.
Beacon calls I$WritLn	A holds 1, X points to the text and Y holds 14.	A, X's meaning and the byte count are the same.
OS-9 resolves path 1	The path leads through the character file manager and terminal driver.	The path leads through RBF and the HD1 disk driver.
Beacon finishes	F$Exit returns the status to its parent shell.	The process exits and the saved file can be opened independently.

The number **1** in the program identifies a slot in *that process's* path table. It does not permanently mean screen or disk. That small indirection explains why a program written by someone who never saw your practice disk can still write a file there.

Catch the error at the right layer

With **/h1/report** still on the practice disk, enter exactly the same redirection and then read the existing file.

```
/h1/beacon >/h1/report
list /h1/report
```

The repeated create reports **ERROR #218**. The listing still prints **beacon report**. The shell needed to open the destination before launching the child. When that open failed, no Beacon instruction ran. A program's carry check after I$WritLn handles a different case, a failure *after* a process has started and requested a write.

Give the line to another program

Can you display only matching lines without changing any source or creating another report on disk? Predict what **grep** will receive, then try:

```
/h1/beacon ! grep report
```

The guest's GREP displayed **: beacon report**. The leading punctuation belongs to the filter's display of the match. The module supplied the same line it wrote to disk. In this run the shell connected Beacon's output to a pipe and the filter read the other end. The filter still had its own output path to the terminal.

```
Ty/La At/Rv: $11 $81
Prog mod, 6809 obj, re-en, R/O

(Term!02)/H1:attr /h1/beacon e
--e-rewr

(Term!02)/H1:/h1/beacon
beacon report

(Term!02)/H1:/h1/beacon >/h1/report

(Term!02)/H1:list /h1/report
beacon report

(Term!02)/H1:/h1/beacon >/h1/report
ERROR #218

(Term!02)/H1:list /h1/report
beacon report

(Term!02)/H1:/h1/beacon ! grep report
: beacon report

(Term!02)/H1:
```

Figure 14: The guest runs Beacon on the terminal, routes it into a file, catches error 218 on a repeated create, reads the intact file and finds the report through a pipeline.

You can now explain a trip that begins at a key and ends in a disk sector. Shell+ interprets the command, opens a path and starts a process. The process requests a service. A file manager interprets that request for this kind of device, and its driver transfers the bytes. Back at Shell+, **list** is a different process opening that same file for reading. For another way to make an OS-9 program take part in this arrangement, try the period C compiler.

Find a command

Appendix A

You remember the job, or you remember the name. Start with either. The spellings below describe the prepared NitrOS-9 Level Two image in VCC Plus. A different boot disk can supply a different shell, command set or edition of a tool.

Start with the job

I want to…	Start here
Find a file or figure out which directory the shell is using	Working and execution directories, then `dir`, `pwd`, `pxd`, `chd`, `chx`
Save several commands and run them together	Shell procedures, then `build`, `list`, `shell`
Send the same text to the screen and a file	Paths and pipes, then `tee`
Compare two versions before replacing a file	Recover a copy, then `copy`, `cmp`, `rename`
See who's using the CPU or holding memory	Processes, modules and address maps
Identify a disk or change a terminal option	Devices, terminals, then `devs`, `dmode`, `tmode`
Run my own program inside OS-9	BASIC09, assembly and the language guide

How to read a command form

Words in italics stand for your own names. Square brackets enclose optional operands. A *device* is a guest name such as /h1. A *path* can also name a file or directory on that device. A *module* is identified by its internal name. For example, `ident /h1/beacon` reads a file, while `link beacon` asks for a module by name. The module chapter explains when that name becomes available.

Shell+ interprets operators before it starts the command. A single > creates a new output file, `!` connects two programs, and & starts a background job. A procedure is text read by a shell. A command can also be built into the shell or supplied by an executable module, including one loaded at boot. Looking only in /DD/CMDS will miss some names the shell accepts.

The compact forms below cover the guest help catalog. A lookup entry identifies an operation. The linked lessons give inputs and visible results for the operations used in the book. For edition-specific switches, use the guest's HELP entry for the named command. Names such as GFX and INKEY belong inside a BASIC09 procedure.

Alphabetical finder A–D

Name and form	What it does
ASM *source* [o=*module*]	Translate assembler source. Read the reported error count before attempting to execute the output.
ATTR *path* [*permissions*]	Inspect or change permission bits. Adding `e` gives the owner permission to execute a file.
BACKUP [*source-device destination-device*]	Duplicate a volume at the device level. The destination contents are replaced. See the working-image recovery exercise for a Mac-side alternative.

Name and form	What it does
BASIC09	Open the programming workspace at its B: prompt.
BINEX *input output*	Convert binary data to Motorola S-record text for transfer. EXBIN performs the reverse conversion.
BUILD *path*	Make a short text file from lines you enter. A blank Return ends entry.
CHD *directory* · CHX *directory*	Change the current data directory or the command execution directory respectively.
CMP *first second*	Compare files byte by byte and report differences.
COBBLER *device* · CONFIG	Construct boot media. Work on a spare image with an independently bootable original.
COPY *source destination*	Duplicate a file under a fresh name.
CPUTYPE · DATE	Report the guest processor or the guest date.
DCHECK *device*	Check the relationships among a disk's allocation map, directories and files.
DEBUG · DED *device*	Inspect machine execution or edit raw disk sectors. A raw write can change several directory entries at once.
DEINIZ *device*	Release a device attachment. INIZ acquires one.
DEL *path* · DELDIR *directory*	Remove a file, or remove a directory tree and its contents.
DEVS	Show the active devices and their drivers and file managers.
DIR [-e] [*path*]	List a directory. The extended listing shows ownership, attributes and sizes.
DIRSORT [*directory*]	Reorder entries in the directory itself. Use a copy if the original order matters.
DISASM [*path*] · DISPLAY *codes*	Decode instructions, or emit character values such as terminal controls.
DMEM [*block*] *offset* [*length*]	Inspect bytes from a guest memory block.
DMODE *device* [*settings*]	Read or change an RBF disk descriptor's settings.
DSAVE [*destination*]	Generate a procedure that copies a directory hierarchy. Inspect the generated commands before running them.
DUMP *file*	Show the offset, hexadecimal bytes and printable characters side by side.

Alphabetical finder E–M

Name and form	What it does
ECHO *words*	Write a line on standard output. The shell can redirect it.
EDIT [*file*] · MINTED [*file*]	Edit text using the editor supplied by the guest. BUILD suffices for all required source entry here.
ERROR *number*	Look up a guest error number. Read it alongside the command that produced it.
EX *module*	Replace the current shell process with the named program.

Name and form	What it does
EXBIN *input output*	Decode an S-record file into binary.
FORMAT *device*	Initialize a filesystem. An existing directory and its data can be lost.
FREE *device*	Report the volume's geometry and available storage.
GFX · GFX2 · GRFDRV	The first two are BASIC09 graphics interfaces. GRFDRV supplies lower-level window drawing services.
GREP *pattern*	Select matching input lines. The pipe exercise supplies its input.
HELP [*topic*]	Consult the help text installed on the guest disk.
IDENT *module-file*	Read module identity, revision, length and CRC information from a file.
INIZ *device*	Attach a device so that the system can keep it initialized.
INKEY	Read a keystroke through a BASIC09 subroutine.
IRQS	Inspect the guest interrupt service table.
KILL *process-ID*	Send an abort signal to the selected process. Use the ID from the current PROCS display.
LINK *module* · UNLINK *module*	Acquire or release a reference to a module already known to the kernel.
LIST *path*	Send a text file's contents to standard output.
LOAD *path*	Bring one or more modules from a file into memory.
LOGIN	Start a login dialogue using the guest account configuration.
MAKDIR *path*	Create a directory entry and its directory file.
MDIR	Enumerate the modules the kernel currently knows.
MEGAREAD	Measure sustained reads from a supplied source.
MERGE *file …*	Send the files' contents in sequence to standard output.
MFREE · MMAP	Report available physical memory or map the system's memory blocks.
MODPATCH *patch-file*	Apply a prepared set of module changes.
MONTYPE · MPI	Inspect or select monitor type, or inquire about a Multi-Pak. These guest utilities have their own hardware assumptions.

Alphabetical finder O–X

Name and form	What it does
OS9GEN *device*	Construct a disk bootstrap. Match the boot track and modules to the target hardware.
PADROM *size file* · PARK *device*	Prepare a padded ROM file, or park heads on a compatible physical drive. VCC Plus VHDs have no mechanical heads.
PMAP · PROC · PROCS	Inspect a process's memory map or the system's process list.

Name and form	What it does
PROMPT *message*	In the supplied help, the PROMPT utility displays a message and waits for a key.
PWD · PXD	Print the full path of the shell's data or execution directory.
REBOOT	Restart the guest. Allow writable media to finish saving first.
RENAME *old-path new-name*	Change the final name of a directory entry.
RUNB *procedure-name*	Run a packed BASIC09 procedure. See the Night procedure.
SAVE *file module …*	Write named resident modules to a file.
SETIME [*date/time*] · SETPR *ID priority*	Set guest time or change the scheduling priority of a process.
SHELL [*procedure*]	Start an interactive child shell or have it interpret a procedure file.
SLEEP *ticks*	Suspend the caller for a specified interval. Used as an identifiable background child in this book.
SMAP	Display a system memory map.
TEE *path*	Copy standard input to standard output and another destination.
TELNET *host port*	Connect a guest terminal session. See the Mac permission dialogue.
TMODE [*options*]	Inspect or change an open terminal path's options.
TOUCH *file*	Update the date associated with a file.
TSMON [*terminal*] · TUNEPORT *device*	Manage waiting terminal sessions or adjust a supported serial port.
VERIFY [*options*]	Check module integrity. Inspect update options before applying them to a file.
WCREATE [*options*] · XMODE *device* [*settings*]	Set up guest windows or inspect/change the defaults in an SCF device descriptor.

Explore the rest of /DD/CMDS

The supplied image contains 398 named directory entries here. A filename alone cannot tell you whether the entry is a shell command, an application, a data file or a bundle of modules. For instance, `DIR`, `DIR.NEW` and `DIR.ORIG` are three different entries. The system help catalog includes names that live elsewhere or are built into a shell. Use the finder above for that catalog. Use the directory below when an unfamiliar name turns up on this particular disk.

The following groups locate additional entries by task. Presence in this inventory means the name was read from the prepared filesystem. An executable module still needs compatible hardware, any companion files and the correct invocation. For an unfamiliar file, use the extended listing to check its type and permissions, and IDENT to inspect a module header. Consult the guest's /SYS/MAN and /SYS/helpmsg for available instructions.

Look under	Additional names on this image
File handling, conversion and disk tools	`ar`, `bfre`, `bfree`, `bigfile`, `cat`, `copy2`, `cp`, `d`, `ddmaker`, `dir.new`, `dir.orig`, `dloadx`, `dloady`, `dname`, `exttran`, `filepick`, `filer`, `files`, `fixtext`, `fmap`, `gcpy`, `ln`, `lof`, `ls`, `mscopy`, `mv`, `pcdos`, `pcsave`, `readdisto`, `rescue`, `rsdos`, `rssave`, `sdc`, `sortdir`, `tar`, `transfer`, `trscopy`, `unzip`, `vdisk_mods`

Look under	Additional names on this image
Text, tabulation, editing and formatting	adj, cut, cuts, dict, diff, fgrep, gsort, lninput, lower, macro, maketxt, more, mroff, nroff, qsort, scred, sled, sort, spell, split, strings, tail, text, tsed, tsedit, txt80x25, ul, uniq, unwords, upper, wc, words
Assembling, compiling, linking and debugging	basecon, c, c.opt, c.pass1, c.pass2, c.prep, cc, cmdgen, crcoff, crcon, ipatch, make, makpatch, modbuster, rlink, rma, sleuth, syscall
Pascal and other language systems	kcbasic, pacos9, Pascal, Pascal_Compiler, PascalE, PascalErrs, PascalN, PascalS, PascalT.MODL, PascalT.PRUN, PCODEF
System, boot, accounts and configuration	adduser, alias, at, bios, bios3, chown, config.desk, config.desk.save, control, cron, date.orig, datetime, drivewire_mods, dtime, expire, fix, fstat, getclock, ide_ff50_mods, ide_ff60_mods, initsub, l2initsub, logname, mkdir, mkopt, multistart, os, os9gen.orig, ostart, printer_mods, profile, reset, setime.orig, setype, shell_21, sysdev, turnkey, user, utilpak1
Windows, graphics, fonts and printing	artist, bpix, cls, cm32vef, cocopr, color, desk, desk.old, desk.patch.scr, fixpix, g320c16, g640c2, g640c4, gclock, gif2vef, gport, gprint, gshell, guib, lpic, lpr, make80, makegshell, makegw, paint, pix, play, rleconv, rleview, scrn, vef2gif, vefio, view, viewgif, window, window.glr4, window.t32, window.t80b, window.t80s, wpix
Communications, mail and file transfer	accdata, area, areacode, autodial, crypt, httpd, inetd, kermit, mail, osterm, pop, postnews, readnews, remctrl, rmail, rnews, supercomm, telecom, uploadx, uucico, uucp, uudecode, uuencode, uuxqt, xmodemup, xymodem
DeskMate, calendar and office software	calendar, dm3, dmalt, dmalt.hlp, dmapps, dmbackup, dmcaln, dmcaln.hlp, dmcards, dmcards.hlp, dmdskini, dmhelp, dmledger, dmledger.hlp, dmload, dmmenu, dmmenu.hlp, dmpaint, dmterm, dmterm.hlp, dmtext, dmtext.hlp, dynacalc, gcal, gcalc, jdate, ledger, month, zone
Games, music and demonstrations	ball, banner, Bounce, cave, caveman, cookie, cursive, demo, des1, des2, des3, des4, des5, des6, dungeon, flight, Fran, ICONS, klondike, koronas, lander, maze, meep, MIDI.dddr, play.orig, playmus, quest, questdvr, questm, rogue, SerMIDI, sound, spals, speech, tank
Other historical applications, scripts and companion files	autoex.count, automode, dmpaint.hlp, dmtim, dw, ed, ezgen, gost, hdlns, hisub, lha, mgt, movsub, pag, pak, pmpts, PROCTAB, qtip, rep, rsb, savsub, sdcdriver, show, solution, Support, Support1, Support2, swread, tree, tspars, UBox3, Um3B, Um3D, Um3E, Um3K, Um3L, Um3M, Um3P, Um3S, Um3T, Um3U, Um3W, UMusE3, vu, xcom9, xlh

For a new system disk, inspect its own command directory and module list. Historical program collections often package multiple editions together. The tested command in a lesson is always identified by the guest and its visible result.

Device and storage atlas

Appendix B

Use this atlas when a name, a sector number and a Mac file all seem to point to the same thing. Each locates an object at a different layer.

Trace a request to its destination

Guest name or handle	OS-9 route on the supplied system	Destination
/dd	Default system disk descriptor through RBF	The boot filesystem supplied on HD0
/h0	H0 through RBF and EmuDsk, drive 0	The HD0 image shown under Virtual Hard Disks
/h1	H1 through RBF and EmuDsk, drive 1	Your writable HD1 image, with an initialized OS-9 filesystem
/Term	Term through SCF and VTIO	The guest terminal displayed in the emulator
A shell pipeline **!**	Pipe manager, with an open write end and read end	Bytes passed from one guest process to the next
Standard output, **1**	Per-process path number, inherited or redirected by the shell	Terminal, file or pipe selected for this run

devs reports relationships among active devices, drivers and managers. **dmode /h1** identifies the disk descriptor and its drive number. **tmode** inspects the current terminal options. The quick-start equipment map pairs the drive window with guest names. The HD1 virtual device serves sectors. It is RBF that interprets them as directory entries and file descriptors.

Decode the station disk

All offsets below are hexadecimal byte offsets *inside the named structure*. Multi-byte integers here are most-significant byte first. The example values come from chapter 5's actual exercise disk.

Structure	Offset and width	Value in the example
Volume description	$00, three bytes, total sectors	05 A0 00 → 368,640 sectors
Volume description	$04, two bytes, allocation map length	16 80 → 5,760 bytes
Volume description	$06, two bytes, sectors per cluster	00 08 → 8
Volume description	$08, three bytes, root descriptor LSN	00 00 18
report directory entry	$00–$1C, name field	72 65 70 6F 72 F4 → **report**. The high bit of its last character marks the end
Same 32-byte entry	$1D, three bytes, descriptor LSN	00 00 28
File descriptor at $28	$00, attributes, followed at $01 by two-byte owner ID	03 → owner read/write. Owner 00 00
File descriptor	$08, one byte, link count	01
File descriptor	$09, four bytes, file length	00 00 00 17 → 23 bytes

Structure	Offset and width	Value in the example
First segment entry	$10, three-byte start, then two-byte sector count	00 00 29 00 07 → seven sectors beginning at $29
File data	$00 from start of the first segment	65 63 68 6F → **echo**

A descriptor can hold more segment entries when a file grows in disjoint runs. Use the declared file length to stop. The descriptor's link count is a filesystem property. A module's in-memory link count has a different lifetime and purpose.

Identify an image before changing it

free /h1 reports the filesystem's volume name, capacity and free space. A copied disk keeps its volume name, so the name alone cannot identify the right backup. Use a known file or a time-specific marker, then confirm the Mac file assigned to the drive. Recover the right copy uses both checks.

For the layout and terminology, see The NitrOS-9 Project, *NitrOS-9 Technical Reference*, revision 0.1 (2004), chapter 5, pp. 48–52. The example sector numbers and file contents in this atlas were measured on the VCC Plus practice disk.

System calls and errors

Appendix C

Beacon takes the same route to the monitor and the disk. Its assembler emits a software interrupt instruction (SWI2) followed by one service byte. NitrOS-9 reads that byte, consults the process's registers, and dispatches the request. A successful write returns to the next instruction in Beacon. An exit ends the process.

The two calls in Beacon

Service	Before the call	After the call	Where you can see it
I$WritLn · $8C	A holds the output path number. X points to the first byte. Y holds the maximum count. The buffer contains a carriage return.	On success carry is clear and Y gives the count written. The line write honors the first carriage return within that limit. On failure carry is set and B identifies the error.	Follow the buffer through the assembler listing and change its destination.
F$Exit · $06	B holds the exit status. Beacon supplies zero after success or the error number returned by the write.	Control passes to the operating system. A waiting parent can receive the child's ID and status.	The shell accepts a new command after Beacon finishes. See the process lesson for the parent's view.

`I$` identifies an I/O request and `F$` a kernel service. Both use the OS-9 calling convention. The mnemonic after the prefix matters: I$Write takes a byte count and can transmit binary data containing carriage returns. I$WritLn has line semantics. For software that reads a line, I$ReadLn takes a path in A, buffer address in X and capacity in Y, and returns the actual count in Y on success. Keep the numeric path separate from the address of the character buffer.

I/O service finder

These are the named I/O requests in this image's system definitions. File managers implement operations appropriate to their devices. For example, seeking to a byte position makes sense on a disk file. A terminal has different status operations. The device chapter shows how OS-9 chooses the file manager.

Service	Request
I$Attach, I$Detach	Acquire or release an attachment to a named device. An open path can keep the device in use.
I$Dup	Get another path number referring to an existing open path.
I$Create, I$Open, I$Close	Create a new entry and open it, open an existing object in an access mode, or release an open path.
I$MakDir, I$ChgDir	Create a directory or change the calling process's working directory for data or execution.
I$Delete, I$DeletX	Delete using the data directory, or specify the mode used to resolve the pathname.
I$Seek	Move the position associated with a random-access file. The next transfer starts there.
I$Read, I$Write	Transfer a counted buffer. Check the returned count as well as carry.
I$ReadLn, I$WritLn	Transfer a line using a bounded buffer. The terminator matters.
I$GetStt, I$SetStt	Inquire about or change a path/device setting. A second function code determines the operation, its registers and whether the manager supports it.

Services encountered by a user program

Use this index to recognize names in source, disassembly and the guest's /DD/DEFS/os9.d. The definition file assigns numbers across several OS editions and extensions. A name in the file does not establish permission or implementation in the running kernel.

Look for	Services and their purpose
Start, replace or finish a process	F$Fork creates a child. F$Chain replaces the current program. F$Wait collects a child termination, and F$Exit supplies its status.
Process identity and scheduling	F$ID identifies the caller. F$SPrior sets an eligible process's priority. F$Sleep waits for ticks, F$SSpd suspends, F$Send delivers a signal, and F$Icpt installs its handler. F$SSWI sets a software-interrupt vector.
Find and release modules	F$Link searches by module name. F$Load reads a file and links a module. F$UnLink releases a reference and F$UnLoad looks up the module to release by name. F$NMLink and F$NMLoad are non-mapping variants.
Measure and request memory	F$Mem adjusts the caller's data area. F$GPrDsc, F$GBlkMp and F$GModDr request copies of process, block-map and module-directory information. F$CpyMem copies across an address-space boundary subject to the kernel's rules.
Process text, names and checksums	F$PErr writes a system error. F$PrsNam parses a pathname component and F$CmpNam compares names. F$CRC updates a cyclic redundancy check. F$SchBit, F$AllBit and F$DelBit search, allocate or clear bits in a map.
Clock and identity changes	F$Time reads the clock and F$STime sets it. F$Alarm and F$TimAlm concern alarms. F$TPS reports ticks per second. F$SUser changes a user identity subject to privilege checks.

Services used inside the operating system

System code also uses SWI2 services. A driver running in system state can manipulate data structures an ordinary application must reach through a public service. Encountering a name here is a reason to identify the caller's privilege and the exact kernel edition before adapting a listing.

Facility	Names to look up and what they connect
Interrupts and scheduling	F$VIRQ, F$IRQ and F$SSvc install service entries. F$IOQu, F$AProc and F$NProc manage I/O waits and runnable processes.
System storage and descriptors	F$SRqMem and F$SRtMem acquire and release system storage. F$Find64, F$All64 and F$Ret64 locate, allocate and release descriptor slots. F$AllPrc, F$DelPrc and F$GProcP concern process descriptors.
Module directory and boot	F$VModul validates a module. F$SLink, F$ELink and F$FModul link or locate directory entries. F$GCMDir compacts the directory. F$IODel releases an I/O module. F$Boot and F$BtMem serve bootstrap code.
RAM and process images	F$AllRAM, F$DelRAM and F$AlHRAM allocate or return RAM. F$AllImg, F$DelImg, F$SetImg, F$MapBlk and F$ClrBlk maintain process address images. F$FreeLB and F$FreeHB find free blocks.
Tasks and DAT translation	F$AllTsk, F$DelTsk, F$SetTsk, F$ResTsk and F$RelTsk maintain task mappings. F$DATLog converts a block and offset into a logical address. F$DATTmp is a historical temporary-image service marked obsolete in the definitions.

Facility	Names to look up and what they connect
Access across mappings	F$Move copies a block of bytes. F$LDAXY, F$LDAXYP, F$LDDDXY, F$LDABX and F$STABX load or store across task mappings. Examine the register contract of the particular entry before using it.
Edition- or hardware-dependent services	F$Debug and F$RegDmp concern debugging. F$ReBoot, F$CRCMod, F$XTime, F$VBlock and F$NVRAM concern reboot, CRC policy, extended time, validating a block of modules, or nonvolatile hardware storage. Treat these as names in the prepared definitions until the specific kernel and driver support is known.

Use an error as evidence

Guest result	What happened here	What you can do
211 · End of file	A reader requested more input after the writer finished or the file ended.	A program consuming a stream can treat this as its normal end condition. Distinguish it from an unexpected read failure.
214 · No permission	An open or operation requested access the target did not grant. The file lab removes read permission from a practice copy.	Check the path, owner, requested access and attributes. Restore the copy's owner-read permission.
216 · Path name not found	A path component failed to resolve. A procedure's data directory can differ from the directory containing the command program.	Inspect the current directory and try the full known data path.
218 · File already exists	Creation of a new output collided with a name already on disk.	List the existing file. Choose a fresh output name when you want to preserve both results.
228 · Process aborted	The parent reported the termination of the Sleep child deliberately signaled in this lab.	Check the process list again. Absence of *that child* and a responsive parent shell are the two expected observations.

Error 043 from RUNB is a language-runtime report. In the Night exercise the successful argument is the packed procedure's internal name, `night`, with the execution directory set to /H1. An error from a language tool need not be a kernel file error. For historical numbers and additional error families consult the bibliography in the language and edition guide.

Choose a language and a delivery format

Appendix D

Start with the program's intended home. A BASIC-hosted binary can expect a running BASIC environment. An OS-9 process receives an entry point, private working storage, paths and operating-system services. Choosing the environment tells you what to build, how to start it, and what a successful return looks like.

Match the artifact to its consumer

What you have	Who reads it	How this book used it
night.b09	BASIC09 in the OS-9 guest	LOAD reads the editable procedure source into the language workspace.
night	The guest's RUNB runtime	BASIC09 PACK produces a named procedure module containing intermediate code. RUNB takes the procedure's name.
beacon.a	The supplied OS-9 assembler	ASM creates a complete 6809 program module. The source contains the module, entry, storage and closing CRC directives.
beacon	OS-9's loader	Shell+ executes it by name or path. It receives the standard paths and returns by F$Exit.
watch.c	The period C compiler in the guest	CC drives several tools. RMA produces relocatable object code and RLINK joins it with the C startup object and library.
A CoCo BASIC program or BASIC-hosted machine-language binary	The corresponding ROM BASIC environment	Use the native BASIC, C and assembly books for their VCC Plus workspaces and direct-run conventions.

The output format is part of an assembler's job. For verification, the bundled LWASM was also run in its OS-9 module output mode on the unchanged Beacon source. Both guest-assembled files matched its 56 output bytes exactly. The reader can produce and execute the complete module using the guest alone.

Compile a small C program in NitrOS-9

This optional exercise uses the **cc version 2.3.0** present on the supplied HD0. Start at Shell+ with the usual **/dd/cmds** execution directory. Use a personal HD1 with fresh names **watch.c** and **watch**. Run:

```
chd /h1
build watch.c
```

Type the Watch source, one line per question-mark prompt, and press Return on an empty line to finish. This short program uses the C dialect accepted by the period compiler. Its untyped **main()** uses the compiler's implicit integer return type. The library routine **puts** writes a string and ends its line. A return value of zero tells the startup code the program completed successfully.

```
main()
{
    puts("night duty");
    return 0;
}
```

Confirm with **list watch.c** that the opening and closing braces appear on their own lines. In the guest keyboard matrix Control-comma produces the opening brace, and Control-period the closing brace. The Mac's natural keyboard layout also maps brace characters to those combinations.

```
cc watch.c -f=/h1/watch
```

Wait for the shell prompt. The explicit output filename is important. The source lives in the data directory, but the linker treats its final output as an executable. Giving it a full HD1 path keeps that file on your practice disk. The compiler created intermediate files as it worked and printed the stages it invoked.

Stage shown on screen	Job
c.prep	Preprocess the C source, including any requested headers or macro expansions.
c.pass1 and **c.pass2**	Translate the C program in successive passes.
c.opt	Optimize the generated assembly.
rma	Assemble a relocatable object whose final addresses can be resolved at link time.
rlink	Combine the object with **/DD/lib/cstart.r** and **/DD/lib/clib.l** to make the executable.

```
/h1/watch
ident /h1/watch
```

Look for **night duty** and a return to Shell+. Both guest processors produced a 1,759-byte module with a good CRC of **$F01AE3** for this exact source and toolchain. Your seven-instruction Beacon supplied its own entry and exit. This C program also carries the startup and library code needed by what you wrote.

```
?

(Term!02)/H1:list watch.c
main()
{
    puts("night duty");
    return 0;
}

(Term!02)/H1:cc watch.c -f=/h1/watch

cc version 2.3.0
'watch.c'
   c.prep:   watch.c
   c.pass1:   ./ctmp3.m -o=./ctmp3.a
   c.pass2:   ./ctmp3.a -o=./ctmp3.i
   c.opt :   ./ctmp3.i ./ctmp3.m
   rma   :   ./ctmp3.m -o=./ctmp3.r
   rlink :   /DD/lib/cstart.r ./ctmp3.r -o=/h1/watch -l=/DD/lib/clib.l

(Term!02)/H1:/h1/watch
night duty

(Term!02)/H1:
```

Figure 15: The running guest lists the C source, shows the compiler passes, assembler and linker, then executes the finished OS-9 command and prints night duty.

Read further

- Microware Systems Corporation. *BASIC09 Programming Language Reference Manual*, revision H (1984). See “Introduction to BASIC09 Programming” and “Program Statements and Structure” for procedures, loops and declarations.
- Microware Systems Corporation. *C Compiler User’s Guide for OS-9* (1983), chapter 1, “The C Compiler System.” The compiler front end and flags above were checked on the supplied image.
- Tandy Corporation. *OS-9 Level Two Development System* (1987), “Relocating Macro Assembler,” chapters 1–2, 5 and 9. Use its discussion of sections, external references and the linkage editor when you move beyond a single module.
- The NitrOS-9 Project. *NitrOS-9 Technical Reference*, revision 0.1 (2004), chapters 3–4 and 8. See the module header layout, standard paths and the system-call interface used in Beacon.

Find your way back into the system

Appendix E

An OS-9 manual from the 1980s can explain a process descriptor in today's emulated CoCo. Its example boot disk may give the terminal a different driver or start a different shell. Use the book to understand the structure, then use the guest to identify the particular parts you have.

Which edition is on the screen?

The prepared image identifies itself as **NitrOS-9/6809 Level 2 V3.3.0** and dates its build August 14, 2015. Its interactive shell identifies itself as **Shell+ v2.2a**. All required programs and practice files in these chapters use this supplied system and your own writable HD1. The guest operating system, shell, BASIC09 and C compiler each have their own versions.

Question	Where to look
Which processor and memory did I configure?	The running VCC Plus machine and its profile. The book's exercises use 512 KB. A 6809 program can also run on the HD6309 in its startup mode. Both runs here used the same 6809 OS image.
Which shell, disks and device drivers am I actually using?	Read the startup banner, check the two mounted hard-disk images in the Mac controls, then consult the device table.
Why does a command in a period manual behave differently?	Compare the manual's shell and utility editions to the command finder and the guest's own help. For example, the second greater-than sign in this Shell+ output syntax selects the error path.
How do I get back to a known boot?	Preserve the bootable system image and restore the personal practice data. For a separate boot experiment, start from a **fresh personal copy** of the built-in profile and confirm it reaches Shell+ before changing it. A working copy of HD0 together with its matching profile gives you a route back to that boot.

Read by question

- **I'm new to OS-9.** Start with Book 8's practice setup. Once you have recovered your captain's log, write the station procedure.
- **Where are my files?** Trace the file through the data and execution directories, then follow its sectors. The storage book explains why the same VHD also holds BASIC volumes.
- **What is the CPU actually doing?** Start with the CoCo 3 MMU lesson, then match a guest process to physical memory. The debugger workbench identifies the Mac controls used to inspect the machine.
- **I want to write a command.** Build the same kind of executable the shell knows how to start. Follow Night in BASIC09, Beacon in assembly, or the guest C build. Book 6 covers the Mac CMOC workbench for BASIC-hosted programs.

Trace a command lookup in a period guide

When you type a command name, where does the shell look after checking the module directory? Choose **Help → Archive Library…**, select **Entire Archive** and search for `OS-9 Level II module directory`. Under Dale Puckett and Peter Dibble's *Complete Rainbow Guide to OS-9 Level II*, open the result for PDF page 79. The

printed page number is 64. It explains how the shell looks in the current execution directory when the requested module is absent from memory. Compare the guide with the Echo module on your practice system.

Choose **Copy Citation** to keep the Archive's original ZIP link, the name of the PDF inside it and the PDF page you opened. If you refer someone to the printed book, include page 64 as well.

Books behind the exercises

- The NitrOS-9 Project. *NitrOS-9 Operating System User's Guide*, version 03.02.01 (2004). §§4.2–4.7, pp. 23–28, cover redirection, concurrent commands and procedures. Chapters 5–6, pp. 30–37, explain process scheduling and startup organization.
- The NitrOS-9 Project. *NitrOS-9 Technical Reference*, revision 0.1 (July 10, 2004). Chapters 1–4 describe the kernel, Level Two memory, module headers and paths. Chapters 5–7 describe the random block, sequential character and pipe file managers. Chapter 8 indexes the service requests. See F$Exit, p. 99, F$Fork, p. 100, F$Wait, p. 128, I$ReadLn, p. 144, and I$WritLn, p. 148.
- The NitrOS-9 Project. *The NitrOS-9 Level 2 Windowing System*, revision 0.1 (June 4, 2004), pp. 7–8, 20–23, 44, 49 and 58. These sections describe creating, selecting, drawing into and closing the graphics window.
- Tandy Corporation and Microware Systems Corporation. *OS-9 Commands Reference* (1986). Compare the original CHD/CHX, DIR, LIST, PWD/PXD and process-command entries with the contemporary guest's command help.
- Tandy Corporation. *OS-9 Level Two Development System* (1987). "Relocating Macro Assembler," chapters 1–2, 5 and 9. See the language guide for BASIC09 and C sources and the build paths used on this system.

The Color Computer Archive preserves historical documentation that informed this book. The lessons, programs and instructional captures here were produced for VCC Plus.

Original manual prose and instructional images © 2026 Brian K. Holman. See artwork credits and the application's Licenses & Notices.

Publication and artwork notices

Original VCC Plus prose and instructional images © 2026 Brian K. Holman. Original component and example notices remain with their respective material.

Except where otherwise identified, original artwork owned by Brian K. Holman is © 2026 Brian K. Holman. All rights reserved.

Permission is granted to use and redistribute this artwork unchanged with VCC Plus, including modified versions of its GPL-licensed software. Other reproduction, modification or redistribution requires prior written permission, except as permitted by applicable law. Third-party materials remain subject to their respective licenses and rights.

The CoCo 3 and CM-8 monitor presentation in VCC Plus's skeuomorphic System view was inspired by SockMaster's "pic of CoCo 3 and CM-8 monitor," published on CoCopedia. VCC Plus's artwork was designed and rendered independently. The remaining hardware illustrations were independently designed using historical photographs and renderings as references.

Read this book online

CMOC license notice · LWTools COPYING notice