

# National Semiconductor Digitalker Digital Voice Selection Software

These are my notes on what I've discovered so far.

## DVSS Programs:

ALIST - list the contents of an archive  
CRCK - checksum calculator  
IBUILD - Digitalker I ROM image builder  
IBURN - ROM image programming for the Starplex  
VLC - vocabulary list compiler  
VLE - vocabulary list editor

## Files:

CRCLIST.CRC - list of file checksums

VERIFY.ROM - preconfigured ROM image that can be burnt to FLASH  
VERIFY.SUB - a script that builds a ROM image  
VERIFY.VOC - source file of words/phrases/sounds to speak  
VERIFY.WRK - some intermediate file maybe

Appendix A - list of all the words supplied in the DVSS library.  
Appendix B - my observations on the verify.\* files and the associated ROM.  
Appendix C - my thoughts on the format of the verify.voc file.  
Appendix D - my walkthrough for creating a custom ROM.

---

A bit of background.

I'm experimenting on a CP/M v2.2 system using boards from <https://smallcomputercentral.com/>.

My setup includes:

- SC108 - Z80 Processor Module for RC2014
- SC110 - Z80 Serial Module for RC2014
- SC129 - Digital I/O for RC2014
- SC145 - Compact Flash Module for RC2014

I have a 512Mb Compactflash card installed. Drive B holds the files from the Digitalker program disk image and Drive C holds the files from the Digitalker archive disk image.

I used the CPM Tools GUI utility found at <https://www.heinpragt-software.com/cpmbox-a-cpm-2-2-emulator/> to extract the contents of the Program Disk and Archive disk .img files.

I then used Grant Searle's Windows Packager Program which I got from <http://www.searle.wales/> - scroll down to the heading "ROM FILES AND CP/M SYSTEM FILES" and download z80sbcfiles.zip from the link that says "DOWNLOAD FILES HERE".

I could then use the packager program to create the text file version of the Digitalker program and archive files.

I transferred the Digitalker program files onto my B drive and the archive files onto my C drive on the CP/M system.

I don't know if it would be an issue putting all the files onto the same drive. Something to experiment with later.

## ALIST.COM – [ list the contents of an archive ]

Typing ALIST ? results in the following:

```
DVSS <alist> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Name:
    alist -- list the contents of an archive
Usage:
    alist [-ln] [-p] [-t] archive [file]
Options:
    '-ln' -- "n" is the listing page length (default=66)
    '-p' -- send output to printer
    '-t' -- terse mode (suppress informational messages)
Arguments:
    'archive' -- the name of the archive to list
    'file' -- the name of the output file to create
```

If I type the following:

```
alist c:stdarc
```

Then the ALIST program dumps out the list of words stored in the STDARC.DAT file. See Appendix A for the output generated from the STDARC file.

## CRCK.COM – [ CRC calculation program ]

Typing CRCK results in the following:

```
++NO FILE NAME SPECIFIED++
```

To use this program:

```
COMMANDS:  CRCK [drive:]<filename.filetype> [F]
```

Examples:

```
CRCK MYFILE.ASM Check only MYFILE.ASM
```

```
CRCK *.ASM      Check all .ASM files
```

```
CRCK *.* F      Check all files and make file  
of results <CRCKLIST.CRC>
```

Performing a CRC check on the ALIST.COM file results in the following output:

```
B>crck alist.com
```

```
CRCK ver 4.3 - 24K Buffer - 01/17/81 RBS  
CTL-S pauses, CTL-C aborts
```

```
ALIST .COM CRC = 4F D0
```

```
DONE
```

```
B>
```

The value (checksum) reported is the same as that detailed in the CRCLIST.CRC file.

## IBUILD.COM – [ Digitalker I ROM image builder ]

Typing IBUILD ? results in the following:

```
DVSS <ibuild> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Name:
    ibuild -- Digitalker I ROM image builder
Usage:
    ibuild [-p] [-v] workfile romfile
Options:
    '-p' -- send output to console and printer
    '-t' -- terse mode (suppress informational messages)
Arguments:
    'workfile' -- the name of an existing workfile
    'romfile' -- the name of the ROM image file to create
```

If I use the work file created from the VLC command (see further on for vlc) and I run IBUILD with the command:

```
ibuild test.wrk test.rom
```

I get the following output:

```
DVSS <ibuild> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Building ROM image...

Workfile <TEST.WRK> contains 13 message(s)

Message < 0>
  Word  Instructions
    1         9
    2         1
    3         7
    4         1
    5         5
    6         1
    7         6
    8         1
    9         7
   10         1
   11         7

Message < 1>
  Word  Instructions
    1         5
    2         1
    3         4
    4         1
    5         6
    6         1
```

7 5  
8 4

Message < 2>

Word	Instructions
1	7
2	2
3	9
4	1
5	9
6	1
7	18
8	1
9	5

Message < 3>

Word	Instructions
1	9
2	1
3	6
4	1
5	6
6	1
7	3
8	1
9	4
10	1
11	3
12	17
13	1
14	3

Message < 4>

Word	Instructions
1	9
2	1
3	6
4	1
5	4
6	1
7	5
8	1
9	3

Message < 10>

Word	Instructions
1	13
2	1
3	5
4	1
5	8
6	1
7	10

Message < 12>

Word	Instructions
1	5
2	1

```
3      5
4      1
5      8
```

```
Writing ROM image to file: <TEST.ROM>
```

```
Message pointers:    13 (26 bytes)
  Instructions:    274 (822 bytes)
Speech data used:  4523 bytes
Speech data reused: 3758 bytes
```

```
Total bytes used:  5371 (33%)
Total bytes left: 11013 (67%)
```

```
Run complete
```

The byte count at the end looks useful in determining when the ROMs are getting full.

It looks like the IBUILD program is working with a pair of ROMs to give a total capacity of 16Kbytes.

## IBURN.COM – [ ROM image programming for the Starplex ]

Typing IBURN ? results in the following:

```
DVSS <iburn> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Name:
    iburn -- ROM image programming for the Starplex
Usage:
    iburn imagefile promtype
Arguments:
    'imagefile' -- the name of an existing ROM image file
    'promtype' -- type of PROM to program (2716 or 2732)
```

“Starplex” was a National Semiconductor development system that appeared to include PROM programmer functionality.

I think this program can be ignored as PROMs will be burnt using a different programmer these days!



## VLC.COM – [ vocabulary list compiler ]

Typing VLC ? results in the following:

```
DVSS <vlc> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Name:
    vlc -- vocabulary list compiler
Usage:
    vlc [-c] [-p] [-t] vocab archive workfile
Options:
    '-c' -- check mode (suppress workfile creation)
    '-p' -- send output to console and printer
    '-t' -- terse mode (suppress informational messages)
Arguments:
    'vocab' -- the name of an existing vocabulary list
    'archive' -- the name of an existing archive
    'workfile' -- the name of the workfile to create
```

If I run VLC with the command:

```
vlc verify.voc c:stdarc test.wrk
```

I get the following output:

```
DVSS <vlc> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Initializing...

Compiling vocabulary list...

Using vocabulary list: <VERIFY.VOC>
Using archive: <C:STDARC>

Message < 0>
  Word  Name
    1    <welcome>
    2    <sil40>
    3    <2>
    4    <sil80>
    5    <d>
    6    <sil20>
    7    <v>
    8    <sil20>
    9    <s>
   10    <sil20>
   11    <s>

Message < 1>
  Word  Name
    1    <enter>
```

```

2    <sil40>
3    <1>
4    <sil20>
5    <million>
6    <sil40>
7    <dollar>
8    <-s.ms1>

Message < 2>
  Word  Name
  1     <warning>
  2     <sil640>
  3     <danger.fue>
  4     <sil160>
  5     <evacuate>
  6     <sil160>
  7     <extreme>
  8     <sil40>
  9     <failure>

Message < 3>
  Word  Name
  1     <get>
  2     <sil40>
  3     <ready>
  4     <sil40>
  5     <4>
  6     <sil40>
  7     <a>
  8     <sil40>
  9     <1>
  10    <sil40>
  11    <nano-.rp>
  12    <second>
  13    <sil40>
  14    <delay.r>

Message < 4>
  Word  Name
  1     <get>
  2     <sil40>
  3     <out>
  4     <sil40>
  5     <of>
  6     <sil40>
  7     <the.r>
  8     <sil20>
  9     <rain>

Message < 10>
  Word  Name
  1     <spell>
  2     <sil40>
  3     <the.r>
  4     <sil40>
  5     <word>
  6     <sil320>
  7     <wednesday>

```

```
Message < 12>
  Word  Name
  1     <that.r>
  2     <sil40>
  3     <is.r>
  4     <sil40>
  5     <correct>

Workfile created: <TEST.WRK>
  Messages: <13>
  Words: <63>

Run complete
```

VLC seems to be ok with the sample file supplied and doesn't report any errors.

## VLE.COM – [ vocabulary list editor ]

Typing VLE ? results in the following:

```
DVSS <vle> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Name:
    vle -- vocabulary list editor
Usage:
    vle [-t] [file]
Options:
    '-t' -- terse mode (suppress informational messages)
Arguments:
    'file' -- the name of a new or existing vocabulary list
```

I assume that this is some sort of text editor that the end user would use to create the required vocabulary file.

There are no clues as to what the actual commands are once the editor is running.

An examination of the binary file in HxD did not reveal any further clues.

If its purpose is to create the .voc file for the vocabulary list compiler, then I think VLE can be ignored and a simple text editor on the PC would suffice.

## CRCLIST.CRC

Viewing this file in Notepad++ (and HxD) indicates that it is an ASCII text file that seems to be padded at the end, to make the file 256 bytes long, with the ASCII character 27 (0x1A) => SUB which appears to equate to CTRL+Z.

For the program disk it contains:

VLE	.COM	CRC = 57 F2
VLC	.COM	CRC = 14 B8
IBUILD	.COM	CRC = FE 22
IBURN	.COM	CRC = 5C 54
ALIST	.COM	CRC = 4F D0
VERIFY	.VOC	CRC = 63 FB
VERIFY	.WRK	CRC = BF F3
VERIFY	.ROM	CRC = D2 03
DVSSBT	.OVR	CRC = 4C 99

For the archive disk it contains:

STDARC	.IDX	CRC = 4E 43
STDARC	.DAT	CRC = 89 94

## Appendix A

This is the output produced by the ALIST program from the command:

```
alist c:stdarc
```

It's actually output in a long list of one word/phrase per line but I've condensed it here into columns for ease of viewing.

DVSS <alist> v1.0

Copyright (C) 1983

National Semiconductor Corporation

-ed.fs1	90	attention	clear
-ed.fs2	a	august.r	close
-ed.fs3	a-.fgt	authorize	close.r
-ed.fs4	a-.mqr	auto	closed
-er.ms1	a-.rqt	available.r	closed.r
-er.ms2	able	average	code
-er.ms3	abort	away	cold
-er.ms4	aborted	b	comma
-er.ms5	ac	back	command.r
-ing.fs1	accumulate	barometric.r	common
-ing.fs2	acknowledge	basement	communication
-ing.ms3	activate	bath	complete
-s.ms1	activating.s	battery	condition
-s.ms2	active	been.r	configuration
-th.ms	activity	before.r	connect.r
-uth.ms	add.r	between.r	continue.s
0	address	black	control
1	adjust.s	blocking	converter
10	after	blue	cool
100	after.p	brake	copy
1000	after.r	budget.s	correct
11	again	budgeting	cost.r
12	air	building	count.r
13	aisle	buoy	cross
14	alarm	busy	customer
15	alarm.r	button	cut.mt
16	alert	buy	cutting
17	all.r	by.r	d
18	alternate.sa	c	danger.fue
19	am(time)	call	data
2	amp	cancel	date
20	ampere	capacitance	day
3	an	car	dc
30	and	case	december.r
4	announcement	caution	decrease.r
40	answer.s	cease	default
5	april.r	celsius	degree
50	arrival	cent	delay.r
6	ask	centi-.fp	demonstration
60	ask.r	centigrade	deposit.r
7	assistance	centimeter	depth
70	astern	change	dial
8	at	channel.r	dial.r
80	at.ft	check.r	dialing.s
9	at.r	circuit	did.r

disable	fuse	keypad	nine
disabled	fused	kilo-.fp	nineteen
divide	g	l	ninety
dollar	gallon	leave.s	no.m
door	gas	left	normal
down	get	less	north
e	go	lesser	not
east	going	level	notice
eight	good.m	lie	november.r
eighteen	gram	light	number
eighty	gray	lime	o
electric	great	limit	o'clock
electricity	greater	line	october.r
eleven	green	link	of
else	group	listen.r	off
emergency	h	load	ohm
enable	half	lock	okay
enabled	have	locked	on
end.r	hello.p	loop	one
enter	help	looped	onward
entry	here	low	open
equal	hertz	lower	operator
error	high	m	optical
evacuate	higher	march.r	optical.m
examine	hire	mark	or
exit	hit	may.r	other.s
exit.r	hold	meg-.rp	out
extreme	home	mega-.rp	over
f	hour	message	over-range
fail	house	message.r	p
failed	hundred	meter	pair
failure	hurt	micro-.rp	pan
far	i	mile	parenthesis
farad	if.r	milli-.mp	pass
fast	in	millimeter	pass.r
faster	in.r	million	passed
february.r	in-.mpt	minus	past
feet	in-.rp	minute	per.r
fifteen	inactive	miss	per-.sp
fifth	inches	missing	percent
fifty	incorporated	model	phone.r
fight	incorrect	modem	phone-number
fire	increase.r	module	pico-.rp
fire.fe	insert.r	monday	place
first	interface	monitor	place.s
five	intruder	more	play
floor	invalid.fa	move.m	please
flow	is.r	move.r	please.p
forty	it	my.m	plus
forward	j	n	pm
four	january.r	nano-.rp	point
fourteen	july.r	near	point.r
friday	june.r	need	pound
from	just.s	next	power
frontal.m	k	next.m	present
fuel	key	night	press.r

pressure	set	taping	tone80.mt
program	seven	target.s	tone800
pull.r	seventeen	tea	tone800.mt
pulse.m	seventy	tear	total
pulses	short	teen	touch.r
push.r	should.m	temperature	tracking.s
put.s	side	temporary.s	traffic
q	sight	ten	transfer.m
quarter	sil10	terminate	trip
r	sil160	terminated	true.r
rain	sil20	test	trunk
range	sil320	than	try
rate	sil40	thank	tuesday
re-.mph	sil5	thank.m	turn
re-.mpl	sil640	thank-you	turn.s
re-.sph	sil80	that.r	twelve
re-.spl	sink	the.r	twenty
reach	site	the-.mr	two
ready	six	the-.rt	type
received.r	sixteen	the-.st	u
receiver	sixty	thee.r	un-.rp1
record.r	slow	thee-.mr	un-.rp2
red	slower	thee-.rt	unable
remove.s	smile	then.r	unattended
repair.r	smoke	there	unit
repeat	sound	therm	unknown
replace.p	south	thermal	unlock
replace.r	space	thermal.m	unlocked
reset	span	third	untrue.r
resistance	spare	thirteen	up
response	speed	thirty	use
restore	spell	this.m	use.r
restored	squad	thousand	used
return.s	squad.m	three	utility
reverse	stair	thursday	v
right	star	tide	voice
ring	start	time.r	volt
ring.r	station	tip	voltage
room	status	today.r	vote
route	steam	today's.r	w
run	stern	tone	wait
running	stop	tone1000	wait.r
s	store	tone1000.mt	waiting
safe	storm	tone10000	waiting.s
saturday	stream	tone10000.mt	wake
second	street	tone12500	wake-up
secure	sub	tone12500.mt	warm
security	subscriber	tone200	warning
see	sunday	tone200.mt	was.s
select	supervisory	tone2000	water
send.r	switch	tone2000.mt	watt
sensor	switched	tone400	wave
sent	system.r	tone400.mt	wear
september.r	t	tone5000	wednesday
sequence.m	tank	tone5000.mt	week.s
service	tape	tone80	welcome



west  
what.r  
will.r  
wind.fnq  
wind.rvg  
wish.r

with.r  
within.r  
word  
work  
working  
x

y  
yellow  
yes  
yes.m  
you  
your.r

z  
zero  
zone

## Appendix B

A quick look at the VERIFY.\* files.

The VERIFY.ROM file can be burnt directly to a flash chip and accessed by Digitalker.

The VERIFY.VOC file is a plain text file that holds the words and phrases that are going to be built into a ROM.

Here's the contents:

```
#
# This is a test vocabulary used to verify DVSS
#
0: welcome sil40 2 sil80 d sil20 v sil20 s sil20 s
1: enter sil40 1 sil20 million sil40 dollar -s.ms1
2: warning sil640 danger.fue sill60 evacuate sill60 \
   extreme sil40 failure
3: get sil40 ready sil40 4 sil40 a sil40 1 sil40 \
   nano-.rp second sil40 delay.r
4: get sil40 out sil40 of sil40 the.r sil20 rain
10: spell sil40 the.r sil40 word sil320 wednesday
12: that.r sil40 is.r sil40 correct
```

When I instruct Digitalker to say word/phrase 0, it speaks "WELCOME TO D V S S".

For word/phrase 1, it speaks "ENTER 1 MILLION DOLLARS".

For word/phrase 2, it speaks "WARNING DANGER EVACUATE EXTREME FAILURE".

For word/phrase 3, it speaks "GET READY FOR A ONE NANO SECOND DELAY".

For word/phrase 4, it speaks "GET OUT OF THE RAIN".

For word/phrase 10, it speaks "SPELL THE WORD WEDNESDAY".

For word/phrase 12, it speaks "THAT IS CORRECT".

## Appendix C

This is what I've discovered so far about the format of the VERIFY.VOC file.

A line starting with a # is a comment line and is ignored.

A line ending with a \ indicates a continuation on the next line.

Each line ends with a CR (ASCII code 13) and an LF (ASCII code 10).

There is a blank line between each entry in the file consisting of just a CR and an LF. I don't know if this is purely cosmetic or a requirement at the moment.

A new ROM word (or phrase) starts with a number and a colon. The number is the same number that you pass to the Digitalker chip when selecting what to speak.

**Note:** Each number must always be larger than the previous number and the numbers don't have to be sequential.

After the colon there is a space and then the name of the sound in the archive file STDARC.DAT. This is the name as reported by the ALIST command.

If you wanted word 0 in your ROM to be the word "busy", then I think the like would look like this:

```
0: busy
```

If you wanted word 5 in your ROM to be the phrase "going up", then there are 3 parts to it – the word "going", a pause and the word "up".

The entries called sil<xxx> would appear to be periods of silence to be used between words.

To add the phrase "going up" as word 1 in your ROM, then I think the like would look like this:

```
1: going sil40 up
```

You should experiment with the various periods of silence to see which makes you phrase sound better.

The first few sounds in the archive appear to be word endings. Looking at the example ROM, the word "dollar" is made into "dollars" by speaking the -s.ms1 sound immediately after the word.

I've not had a chance to discover the difference between -s.ms1 and -s.ms2 yet.

There are also word endings to convert words like "lock" into "locked" and "close" into "closing".

The text file does need to be padded with the ASCII character 27 (0x1A) so that the file is multiples of 256 bytes long.

Running the VLC command on an unpadded file causes VLC to report errors.

## Appendix D

Here's my walkthrough for creating a custom ROM.

### Step 1 – create the list of words/phrases

I started with a plain text file and using Notepad++ on Win10 I created my voice list which looked like this:

```
#  
# This is a test vocabulary  
#  
  
0: zero  
  
1: one  
  
2: two  
  
3: three  
  
4: four  
  
5: five  
  
6: 1  
  
7: 2  
  
8: 3  
  
9: 4  
  
10: 5  
  
11: close  
  
12: close -ing.fs1  
  
13: close -ing.fs2  
  
14: close -ing.ms3  
  
15: the.r sil40 customer sil40 is.r sil40 correct
```

### Step 2 – pad out the text file

I used the hex editor HxD to insert the required number of bytes at the end of the text file so that the overall file size was a multiple of 256 bytes.

In HxD, the file looked like this:

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	23	0D	0A	23	20	54	68	69	73	20	69	73	20	61	20	74	#..# This is a t
00000010	65	73	74	20	76	6F	63	61	62	75	6C	61	72	79	0D	0A	est vocabulary..
00000020	23	0D	0A	0D	0A	30	3A	20	7A	65	72	6F	0D	0A	0D	0A	#....0: zero....
00000030	31	3A	20	6F	6E	65	0D	0A	0D	0A	32	3A	20	74	77	6F	1: one....2: two
00000040	0D	0A	0D	0A	33	3A	20	74	68	72	65	65	0D	0A	0D	0A	0....3: three....
00000050	34	3A	20	66	6F	75	72	0D	0A	0D	0A	35	3A	20	66	69	4: four....5: fi
00000060	76	65	0D	0A	0D	0A	36	3A	20	31	0D	0A	0D	0A	37	3A	ve....6: 1....7:
00000070	20	32	0D	0A	0D	0A	38	3A	20	33	0D	0A	0D	0A	39	3A	2....8: 3....9:
00000080	20	34	0D	0A	0D	0A	31	30	3A	20	35	0D	0A	0D	0A	31	4....10: 5....1
00000090	31	3A	20	63	6C	6F	73	65	0D	0A	0D	0A	31	32	3A	20	1: close....12:
000000A0	63	6C	6F	73	65	20	2D	69	6E	67	2E	66	73	31	0D	0A	close -ing.fs1..
000000B0	0D	0A	31	33	3A	20	63	6C	6F	73	65	20	2D	69	6E	67	..13: close -ing
000000C0	2E	66	73	32	0D	0A	0D	0A	31	34	3A	20	63	6C	6F	73	.fs2....14: clos
000000D0	65	20	2D	69	6E	67	2E	6D	73	33	0D	0A	0D	0A	31	35	e -ing.ms3....15
000000E0	3A	20	74	68	65	2E	72	20	73	69	6C	34	30	20	63	75	: the.r sil40 cu
000000F0	73	74	6F	6D	65	72	20	73	69	6C	34	30	20	69	73	2E	stomer sil40 is.
00000100	72	20	73	69	6C	34	30	20	63	6F	72	72	65	63	74	0D	r sil40 correct.
00000110	0A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000120	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000130	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000140	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000150	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000160	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000170	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000180	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
00000190	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
000001A0	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
000001B0	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
000001C0	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
000001D0	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
000001E0	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....
000001F0	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	1A	.....

### Step 3 – transfer the file to my CP/M system

I'm new to CP/M so this is probably the wrong way to do this!

I used Grant Searle's packager program to create the text file version of the test1.voc file and then pasted it into my CP/M machines TeraTerm window. This effectively transferred the padded test1.voc file onto my B: drive on my CP/M machine.

### Step 4 – compile the list

On my CP/M system I typed:

```
vlc test1.voc c:stdarc test1.wrk
```

That produced the following output:

```
DVSS <vlc> v1.0
Copyright (C) 1983
National Semiconductor Corporation

Initializing...

Compiling vocabulary list...

Using vocabulary list: <TEST1.VOC>
Using archive: <C:STDARC>

Message < 0>
  Word  Name
    1   <zero>

Message < 1>
  Word  Name
    1   <one>

Message < 2>
  Word  Name
    1   <two>

Message < 3>
  Word  Name
    1   <three>

Message < 4>
  Word  Name
    1   <four>

Message < 5>
  Word  Name
    1   <five>

Message < 6>
  Word  Name
    1   <1>

Message < 7>
  Word  Name
    1   <2>

Message < 8>
  Word  Name
```

```
1      <3>

Message < 9>
  Word  Name
  1      <4>

Message < 10>
  Word  Name
  1      <5>

Message < 11>
  Word  Name
  1      <close>

Message < 12>
  Word  Name
  1      <close>
  2      <-ing.fs1>

Message < 13>
  Word  Name
  1      <close>
  2      <-ing.fs2>

Message < 14>
  Word  Name
  1      <close>
  2      <-ing.ms3>

Message < 15>
  Word  Name
  1      <the.r>
  2      <sil40>
  3      <customer>
  4      <sil40>
  5      <is.r>
  6      <sil40>
  7      <correct>

Workfile created: <TEST1.WRK>
  Messages: <16>
  Words: <25>

Run complete
```

### Step 5 – build the ROM

On my CP/M system I typed:

```
ibuild test1.wrk test1.rom
```

That produced the following output:



```
DVSS <ibuild> v1.0
Copyright (C) 1983
National Semiconductor Corporation
```

```
Building ROM image...
```

```
Workfile <TEST1.WRK> contains 16 message(s)
```

```
Message < 0>
  Word  Instructions
    1      7
```

```
Message < 1>
  Word  Instructions
    1      4
```

```
Message < 2>
  Word  Instructions
    1      7
```

```
Message < 3>
  Word  Instructions
    1      4
```

```
Message < 4>
  Word  Instructions
    1      6
```

```
Message < 5>
  Word  Instructions
    1      4
```

```
Message < 6>
  Word  Instructions
    1      4
```

```
Message < 7>
  Word  Instructions
    1      7
```

```
Message < 8>
  Word  Instructions
    1      4
```

```
Message < 9>
  Word  Instructions
    1      6
```

```
Message < 10>
  Word  Instructions
    1      4
```

```
Message < 11>
  Word  Instructions
    1      8
```

```

Message < 12>
  Word   Instructions
    1         8
    2         4

Message < 13>
  Word   Instructions
    1         8
    2         2

Message < 14>
  Word   Instructions
    1         8
    2         1

Message < 15>
  Word   Instructions
    1         5
    2         1
    3        11
    4         1
    5         5
    6         1
    7         8

Writing ROM image to file: <TEST1.ROM>

  Message pointers:    16 (32 bytes)
    Instructions:    129 (387 bytes)
  Speech data used:   2262 bytes
Speech data reused:   3376 bytes

  Total bytes used:    2681 (16%)
  Total bytes left: 13703 (84%)

Run complete

```

### Step 6 – get the ROM file back onto the PC

I'm still a beginner with CP/M so this may not be the best way to achieve this.

I got a file called UPLOAD.COM from  
<https://github.com/RC2014Z80/RC2014/tree/master/CPM/UPLOAD.COM>  
 And transferred the pre-made UPLOAD.PKG file to my CP/M system.

I then typed:

```
a:upload test1.rom
```

That generates a file in ASCII format that I could then copy from RealTerm and paste into the web page <https://rc2014.co.uk/fileunpackage/>

That then recreated the binary ROM image which I was able to program into a flash chip.

### Step 7 – testing

I'm using my Arduino Digitalker shield for this bit but each of us will have our own way of getting Digitalker to speak words from the ROM(s).

If I tell the Digitalker chip to play back words 0, 1, 2, 3, 4 and 5 then it says the words zero, one, two, three, four and five.

If I tell the Digitalker chip to play back words 6, 7, 8, 9 and 10 then it says the words one, two, three, four and five.

So, constructing a custom ROM with either the numbers 1, 2, 3 etc generates the same as using the words one, two, three etc.

Playing back word 11 results in the word "close" and playing back words 12, 13 and 14 results in 3 variations of trying to say the word "closing".

Playing back word 15 results in the phrase "THE CUSTOMER IS CORRECT" being spoken.

There's obviously some tweaking to be done to discover more about the various word endings etc, but that's my walkthrough for creating a custom ROM.

I've yet to see what happens if I create a set of words that exceeds 16Kbytes.