T3X/0 LANGUAGE CHEAT SHEET

PROGRAMS

! *comment*

imports

declarations

```
DO
 declarations
 statements
END
```

IMPORTS

```
USE module;
USE module: alias;
```

DECLARATIONS

```
VAR name, ... ;
```

! vector
```
VAR name[size], ... ;
```

! byte vector
```
VAR name::size, ... ;
```

```
CONST name = cvalue, ... ;
```

! enumerate offsets
```
STRUCT name = name, ... ;
```

! function
```
name(name, ...) statement
```

! forward declaration
```
DECL name(integer), ... ;
```

MODULES

```
MODULE name;
 declarations
END
```

```
MODULE name;
  declarations
  DO
    statements
  END
END
```

PUBLIC *declaration*
is public, others are local.

Module declarations are
VAR, **CONST**, **STRUCT**,
and functions.

STATEMENTS

! assignments
```
name := expression;
name[expr] ... := expr;
name::factor := expr;
name::(expr) ... := expr;
```

The last subscript in a chain
of `[]` may be `::`.

STATEMENTS

! function call
```
name(expr, ...);
```

```
IF (expression) statement
```

```
IE (expression)
  statement
ELSE
  statement
```

```
WHILE (expression)
  statement
```

```
FOR (name=expr, expr)
  statement
```

```
FOR (n=x, x, cvalue)
  statement
```

! count from 1 to 100
```
FOR (i=1, 101)...
```

! count from 100 to 1
```
FOR (i=100, 0, -1)...
```

! re-enter loop
```
LOOP;
```

! exit loop
```
LEAVE;
```

! return from function
```
RETURN expression;
```

! return zero
```
RETURN;
```

! exit program
```
HALT;
HALT cvalue;
```

! compound statement
```
DO
  declarations
  statements
END
```

Local declarations are
VAR, **CONST**, and **STRUCT**.

CVALUES

Evaluate at compile time.

constant
– constant
constant **+** constant
constant ***** constant
constant **|** constant

ESCAPE SEQUENCES

\a BEL	**\n** LF	**\t** HT
\b BS	**\q** '"'	**\v** VT
\e ESC	**\r** CR	**** '\'
\f FF	**\s** ' '	

OPERATORS

Op.	P	A	Desc.
`x[y]`	9	L	y'th word of x
`x::y`	9	R	y'th byte of x
`-x`	8	-	minus x
`~x`	8	-	bitwise inverse
`\x`	8	-	logical not
`@x`	8	-	address of x
`x*y`	7	L	product
`x/y`	7	L	trunc. quotient
`x mod y`	7	L	remainder
`x+y`	6	L	sum
`x-y`	6	L	difference
`x&y`	5	L	bitwise and
`x|y`	5	L	bitwise or
`x^y`	5	L	bitwise xor
`x<<y`	5	L	left shift
`x>>y`	5	L	right shift
`x<y`	4	L	less-than
`x>y`	4	L	greater-than
`x<=y`	4	L	less/equal
`x>=y`	4	L	greater/equal
`x=y`	3	L	equal
`x\=y`	3	L	not equal
`x/\y`	2	L	logical and
`x\/y`	1	L	logical or
`x->y:z`	0	R	if-then-else

! unsigned operators:
`.*` `./` `.<` `.>` `.<=` `.>=`
MOD is always unsigned.

FACTORS

`[a-z_0-9]+`	name
`[0-9]+`	integer
`0x[0-9]+`	hex integer
`%integer`	negative int
`'c'`	character
`"text"`	string

0 is false, non-zero is true.
%1 (**-1**) is canonical truth.

TABLES (FACTORS)

! vectors of bytes
```
PACKED [ cvalue, ... ];
PACKED [ string, ... ];
```

! general vector
```
[ integer, string, @global, ... ]
```

! nested table
```
[[1,0],[0,1], ... ]
```

! table with dynamic elements
```
[(expr, expr, ... )]
```

T3X/0 LIBRARY CHEAT SHEET 1/2

T3X CORE MODULE

```
USE t3x: t;
```

t.bpw()
Return bytes per machine word.

t.break(0/1)
t.break(@v)
Intercept keyboard break.
V = semaphore; 0 = reset;
1 = check for break.

t.close(fd)
Close file descriptor *fd*.

t.create(path)
Create file, return file descriptor.

t.getarg(n, b, k)
Copy first *k-1* characters of *n*'th command line argument to *b*. Return number of chars copied.

t.memcomp(b1, b2, k)
Compare byte vectors of size *k*. Return difference.

t.memcopy(d, s, k)
Copy *k* bytes from *s* to *d*.

t.memfill(b, c, k)
Fill *k* bytes at *b* with *c*.

t.memscan(b, c, k)
Return offset of first *c* in first *k* bytes of *b* (or %1).

t.newline(b)
Fill *b* with newline sequence.

t.open(path, mode)
Open *path* in given *mode*.
Modes: T3X.RDONLY, WRONLY, RDWR, APPND

t.read(fd, b, k)
Read *k* bytes from *fd* into *b*.
Return number of bytes read.

t.remove(path)
Delete *path*. %1 = error.

t.rename(old, new)
Rename *old* as *new*. %1 = error.

t.seek(fd, pos, how)
Move file pointer of *fd* to *pos*. How: T3X.SEEK_SET, SEEK_FWD, SEEK_END, SEEK_BCK.

t.write(fd, b, k)
Write *k* bytes from *b* to *fd*.
Return number of bytes written.

t.intr86(int, regs)
t.bdos(c, ade)
t.bdoshl(c, ade)
System-specific functions.

CHARACTER FUNCTIONS

```
USE char: ch;
```

ch.alpha(c)
ch.ascii(c)
ch.cntrl(c)
ch.digit(c)
ch.lower(c)
ch.space(c)
ch.upper(c)
Check character properties.

ch.lcase(c)
ch.ucase(c)
Convert case.

ch.value(c)
Value of decimal digit.

CONSOLE INTERFACE

```
USE t3x: t;
USE console: con;
```

con.calibrate(n)
Calibrate delay loop on DOS and CP/M.

con.clreol()
Clear to end of line

con.clrscr()
Clear screen.

con.csroff()
con.csron()
Cursor on/off. May be a null operation on some systems.

con.getkey()
Wait for key, return keycode.

con.move(x, y)
Move cursor to (x,y). Home position is (0,0).

con.pollkey()
Check for key, return keycode or zero.

con.refresh()
Refresh screen. Also done by **getkey**, **pollkey**, and **wait**.

con.rscroll()
Scroll screen down.

con.scroll()
Scroll screen up.

con.setup(data)
Initialize console module. On CP/M supply TTY control strings. On DOS and Unix use *data*=0.

con.shutdown()
Shut down console module.

con.standout()
Enter standout/highlight mode. May be a null operation on some terminals.

con.normal()
Exit standout/highlight mode.

con.wait(n)
Wait for *n*/m seconds, where m=1000 on Unix and is set up using **con.calibrate** on other systems.

con.wrch(c)
Write character.

con.writes(s)
Write string.

BDOS INTERFACE

CP/M BDOS interface.
See manual.

HOST IDENTIFICATION

```
USE host;
```

HOST.CPM
HOST.DOS
HOST.TCVM
HOST.UNIX
Exactly one constant is true on every host system.

SIMPLE I/O

```
USE t3x: t;
USE string: str;
USE io;
```

io.fnl(fd)
Write newline sequence to file descriptor *fd*.

io.fwrch(fd, c)
Write character *c* to file descritor.

io.fwriteln(fd, s)
Write string *s* and newline sequence to file descriptor.

io.fwrites(fd, s)
Write string *s* to file descriptor.

io.inpuf(fd)
Select input file descriptor for `io.rdch` and `io.readln`.

io.nl()
Write newline sequence.

io.rdch()
Read single character; buffered read, see `io.input`.

io.readln(b, n)
Read line, buffered read.

T3X/0 LIBRARY CHEAT SHEET 2/2

SIMPLE I/O

Continued.

io.wrch(c)
Write character *c*.

io.writeln(s)
Write string *s* followed by newline sequence.

io.writes(s)
Write string *s*.

I/O STREAMS

See manual.

DYNAMIC MEMORY

```
USE t3x: t;
USE memory: mem;
```

mem.alloc(k)
Allocate block of *k* bytes. Return pointer or zero.

mem.free(b)
Free previously allocated block *b*.

mem.init(p, z)
Initialize memory pool *p* or size

mem.walk(b, @z, @u)
Walk chain of blocks. Start with *b*=0. Return pointer to next block and 0 at end. *Z* = size, *u* = used.

RANDOM NUMBERS

```
USE rng;
```

rng.seed(n)
Seed the RNG.

rng.rand(k)
Generate random number x in the range $0 \leq x < k$.

rng.reseed(n)
Add more entropy.

SHELL SORT

```
USE shellsrt: sort;
```

sort.sort(v, k, @p)
Sort vector *v* of length *k*. Procedure *p* returns difference between two elements.

STRINGS

```
USE t3x: t;
USE string: str;
```

STRING.MAXLEN
Maximum length of a string

str.aton(s, r, @p)
Convert ASCII to number; *s*=string, *r*=radix, *p*=end pointer or zero.

str.comp(a, b)
Difference between *a* and *b*.

str.copy(d, s)
Copy *s* to *d*.

str.downcase(s)
Convert to lower case.

str.find(a, b)
Return offset of *b* in *a* or %1.

str.length(s)
Length of string.

str.ncopy(n, a, b)
Copy *n*-1 characters max. from *b* to *a*, append NUL.

str.ntoa(n, r)
Convert number to ASCII, return internal buffer. *N*=number *r*=radix. Use negative *r* for signed *n*.

str.rscan(s, c)
Return offset of rightmost *c* in *s* or %1.

str.scan(s, c)
Return offset of leftmost *c* in *s* or %1.

str.upcase(s)
Convert to upper case.

STRING COMPAT

T3X r7 compatibility string library. See manual.

STRING FORMATTER

See manual.

STRING PARSER

See manual.

SYSTEM FUNCTIONS

DOS and Unix only.

In general a return value of zero means success, a value below zero indicates failure.

```
USE t3x: t;
USE system: sys;
```

sys.chdir(s)
Change working directory.

sys.closedir()
End reading directory.

sys.dup(fd)
Duplicate file descriptor.

sys.dup2(fd, fd2)
Duplicate file desciptor *fd* into *fd2*.

sys.getdir(b, k)
Read up to *k-1* chars of current working directory name into *b*.

sys.getfsize(s, sb)
Fill buffer *sb* with file size; Format:
sb::0 = size / 256^3
sb::1 = size / 256^2
sb::2 = size / 256
sb::3 = size mod 256

sys.getftime(s, tb)
Fill buffer *tb* with file time; for time format, see **sys.gettime**.

sys.gettime(tb)
Fill buffer *tb* with system time; Format: "CYMDhmsH"; C=century; H=second/100

sys.isdir(s)
Return %1, if *s* is a directory.

sys.mkdir(s)
Create directory.

sys.opendir()
Open current directory for reading.

sys.readdir(b, k)
Read up to *k-1* chars of next directory name into *b*. Used after **sys.opendir**.

sys.ready(fd, ms)
Wait *ms* milliseconds for *fd* to become readable without blocking. Return %1 for success. On DOS *ms* will be ignored.

sys.rmdir(s)
Remove directory.

sys.spawn(p, a)
Run subprocess *p* with command line arguments *a* (null-terminated vector). Return exit status or %1.