"User" variables in the Pocket Forth dictionary. Offsets are bytes from "tib".
These are similar to, but not exactly like, Forth's user variables.

Name	Offset	Description
TermBuf	0	the terminal input buffer
IntA7	84	the initial value of the system's stack pointer
Rzero	88	the address of the bottom of the return stack
UFlow	92	the parameter stack's underflow buffer address
Szero	96	the address of the bottom of the parameter stack
Expand	100	the address of the expand routine in the CODE1 or DRVR
FreePt	104	the relative address of the end of the dictionary at startup
FreeSz	106	the amount of free space at startup
DictPt	108	the name address of the last word in the dictionary
NBase	110	the value of the current number base at startup
Held	112	the address for the next character in a numeral
DoesAdr	114	the parameter from a created word for "does>"
fcolon	118	a flag indicating the interpreters state (see "cstate")
fimmed	119	a flag indicating an immediate word
fneg	120	a flag for "number" indicating a negative number
fint	121	a flag indicating input source (see "cblk")
fmacro	122	a flag indicating macro compiling mode (see macro, mcompile)
fbit5	123	a flag indicating bit 5 mode (not implemented)

Pocket Forth Glossary

! (n addr --) say: "store" <standard> Store value n at the relative address, addr.

!PEN (h v --) say: "store pen" Move the graphics pen to the coordinates on the stack.

(dval -- dquotient) say: "sharp" <standard> Convert one digit of a numeral represented by the dvalue, by dividing the value by the numeric base. Use between "greater-than-sharp" and "sharp-less-than".

#> (dval -- addr len) say: "sharp-less-than" <standard> Leave the address and length of a string representing the formatted dvalue. Use with #, "sharps", "sharp", "hold" and "greater-than-sharp".

#S (dval -- 0 0) say: "sharps" <standard> Convert all of the digits of the dvalue according to the current numeric base. Use between "greater-than-sharp" and "sharp-less-than".

' (-- addr) say: "tick" <standard> Return the relative address of the next word from the input stream.

((--) say: "parenthesis" <standard> Begin a comment. A right parenthesis ends the comment.

(.") (--) say: "paren-dot-quote" <standard> This is the runtime word compiled into the dictionary by "dot-quote". "Paren-dot-quot prints the string data which follows it in the dictionary.

(DO) (limit index --) say: "paren-do" <standard> This is the runtime word compiled into the dictionary by "do". "Paren-do" begins loop.

* (n1 n2 -- n1*n2) say: "star" or "times" <standard> Multiply n1 by n2, and leave the 16 bit result on the stack.

*/ (n1 n2 n3 -- [n1*n2]/n3) say: "star-slash" <standard> Using a 32 bit intermediate, "star-slash" puts the scaled result on the sta

+ (n1 n2 -- n1+n2) say: "plus" <standard> Leave the result of n1 plus n2 on the stack.

+! (n addr --) say: "plus-store" <standard> Add the value, n, to the value found at the relative address, addr.

+LOOP (n --) say: "plus-loop" <standard> Used inside of a colon definition, with "do". Increment a loop index by the value n, ar
branch to the beginning of the loop if index is less than limit. (see "loop" and "do")

+MD (offset -- addr) say: "plus-em-dee" Calculate the relative address of an unnamed variable from an offset on the stack.

, (n --) say: "comma" <standard> Write and enclose the value, 'n' into the dictionary.

,$ (--) say: "comma-dollar" Compile a hex number from the input stream into the dictionary. Used to compile traps and machin
code. ,$ is immediate.

- (n1 n2 -- n1-n2) say: "minus" <standard> Leave n1 minus n2 on the stack.

--> (--) say: "load next file" <almost standard> Take a filename from the input stream, and load the file from the disk. If no disk
path is specified, the default is used. When the file is completely loaded, the previous the previous input source is restored
NOTE: File, folder or disk names used must contain no spaces and the complete path must be specified.

-TO (h v --) say: "line to" Draw a line and move the pen to the coordinates on the stack.

-TRAILING (addr count -- addr count') say: "dash-trailing" <standard> Assuming string data on the stack, adjust the count to
eliminate trailing blanks.

. (n --) say: "dot" <standard> Print the value on the stack according to the current number base. (see "base")

." (--) say: "dot-quote" <standard> Used inside of a colon definition to compile "paren-dot-quote" and the string data into the
dictionary. The string data follows "dot-quote" and is delimited by another quote.

.OK (--) say: "dot-oh-kay" Print 'ok' on the screen, indicating that the interpreter ready for input.

/ (n1 n2 -- quotient) say: "slash" <standard> Divide n1 by n2 and leave the 16 bit quotient on the stack.

/MOD (n1 n2 -- quotient remainder) say: "slash-mod" <standard> Divide n1 by n2 and leave the 16 bit quotient and 16 bit
remainder on the stack.

0 (-- 0) say: "zero" <standard> Leave a zero on the stack.

0< (n -- flag) say: "zero-less-than" <standard> Leave a negative one on the stack if n is less than zero. Otherwise leave a zero.

0= (n -- flag) say: "zero-equal" <standard> Leave a true flag on the stack if n is zero. Otherwise leave a zero on the stack.

0> (n -- flag) say: "zero-more-than" or "zero-greater-than" <standard> Leave a true flag on the stack if n is greater than zero. Otherwise leave a zero.

1+ (n -- n+1) say: "one-plus" <standard> Add one to the value, n on the stack. Leave the result on the stack.

1- (n -- n-1) say: "one-minus" <standard> Subtract one from the value on the stack. Leave the result on the stack.

2! (d addr --) say: "two-store" <standard> Store the 32 bit number, d, at the relative address, addr.

2+ (n -- n+2) say: "two-plus" <standard> Add two to the value on the stack, leaving the result on the stack.

2* (n -- n*2) say: "two-star" or "two-times" <standard> Double the value on the stack.

2/ (n -- n/2) say: "two-slash" <standard> Halve the value on the stack, truncating to an integer.

2>R (d --) (rstack: -- d) say: "two to are" Put a 32 bit number on the return stack. Use within colon definitions.

2@ (addr -- d) say: "two at" <standard> Fetch a 32 bit number from relative address on the stack.

2CONSTANT (compile: [d --] run: [-- d]) <standard> Create a 32 bit constant.

2DROP (d1 --) say: "two drop" <standard> Drop a 32 bit number.

2DUP (n1 n2 -- n1 n2 n1 n2) say: "two-dupe" <standard> Duplicate the top two values on the stack, and leave all four values on the stack.

2OVER (d1 d2 -- d1 d2 d1) say: "two-over" <standard> Duplicate the 32 bit second number to the top of the stack.

2R> (-- d) (rstack: d --) say: "two-are-from" Get a 32 bit number from the return stack. Use only within a colon definition.

2ROT (d1 d2 d3 -- d2 d3 d1) say: "two-rote" <standard> Rotate the top three 32 bit numbers.

2SWAP (n1 n2 n3 n4 -- n3 n4 n1 n2) say: "two-swap" <standard> Reverse the order of the top two and the second two values on the stack.

2VARIABLE (compile: [--] run: [-- addr]) <standard> Create a 32 bit variable. See change to "variable", below.

: (--) say: "colon" <standard> "Colon" creates a new word, called a colon definition. The token following "colon" is the name of the new word. Non-immediate words following the name are compiled into the definition until a "semi-colon" is reached.

; (--) say: "semi-colon" <standard> The "semi-colon" ends a colon definition and compiles a machine language return instructio

< (n1 n2 -- flag) say: "less-than" <standard> Leave a true flag on the stack if n1 is less than n2. Otherwise leave a zero.

<# (--) say: "greater-than-sharp" <standard> Set up for number conversion by clearing the pad, and setting 'held' to "pad"-1.

= (n1 n2 -- flag) say: "equal" <standard> Leave a true flag on the stack if n1 is the same as n2. Otherwise leave a zero.

> (n1 n2 -- flag) say: "greater-than" <standard> Leave a true flag on the stack if n1 is more than n2. Otherwise leave a zero.

>ABS (addr16 -- daddr32) say: "to-abs" Convert a relative address on the stack to a double number absolute address. (see ">rel")

>LINK (addr -- link.addr) say: "to-link" <standard> Return the relative address of the link field of the word whose address is on stack.

>NAME (addr -- name.addr) say: "to-name" <standard> Leave the relative address of the name field of the word whose addres on the stack.

>R (n --) return stack: (-- n) say: "to-are" <standard> Remove a value from the parameter stack and place it on the return stac (see "are-from")

>REL (daddr32 -- addr16) say: "to-rel" Convert a double number absolute address on the stack to a relative address.

?BUTTON (-- flag) say: "question-button" The flag is true if the mouse button is down, false if up.

?DUP (n -- n n OR n [if n=0]) say: "question-dupe" <standard> Duplicate the value on the stack only if it is not zero.

?STACK (? --) say: "question-stack" Print a warning, '*?', if stack underflow has occurred.

?TERMINAL (-- flag) say: "question-terminal" <standard> Leave a true flag if a key has been pressed. In the application version events are handled by "?terminal".

@ (addr -- n) say: "at" (some people say: "fetch") <standard> Leave the value found at the relative address on the stack. The address must be even.

@MOUSE (-- h v) say: "at-mouse" Get the coordinates of the mouse pointer.

@PEN (-- h v) say: "at-pen" Get the coordinates of the graphics pen pen.

A>R (addr -- : -- dabs.addr) say: "a-to-are" Convert an address to absolute, and put it on the return stack. This word is used for system trap setup. Use only within a colon definition.

ABORT (--) <standard> Stop execution, print a warning, '?' and return to the interpreter loop.

AGAIN while compiling: (addr --) while executing: (--) <standard> Used to compile an unconditional branch to a relative addr
left by "begin". (see "begin" and "back") "Again" is an immediate word.

ALLOT (n --) <standard> Allocate and enclose n bytes in the dictionary. If n is odd, n+1 bytes will be allocated. The value of th
bytes is undefined at compile time.

AND (n1 n2 -- n1ANDn2) <standard> Leave the result of n1 AND n2 on the stack. The value is computed bitwise.

BACK while compiling: (addr --) no execution behavior <standard> Compiles the difference between the current compilation
address and the address on the stack. "Back" is used by "again", "repeat" and "until". (also see "begin" and "while")

BASE (-- addr) <standard> "Base" leaves the address of a variable containing the current number base.

BEEP (--) Causes the speaker to beep at the current volume.

BEGIN (--) <standard> "Begin" starts a conditional or unconditional loop in the following manner:
BEGIN … (-- flag) WHILE … REPEAT,
BEGIN … (-- flag) UNTIL and
BEGIN … AGAIN
"Begin" is an immediate word and is used within a colon definition. (see "while", "again", "repeat" and "until")

C! (c addr --) say: "sea-store" <standard> Store the 8 bit value at the relative address (even or odd).

C@ (addr -- c) say: "sea-at" <standard> Retrieve the 8 bit value found at the address (even or odd) to the stack.

CBLK (-- addr) say: "sea-bee-el-kay" Returns a relative address which contains a byte value. If the value is 128, then the
interpreter looks for input from the keyboard. A zero value causes text to be interpreted from the file stack.

CMOVE (addr1 addr2 n --) say: "sea-move" <standard> Moves n bytes from addr1 to addr2.

COMPILE (addr --) "Compile" writes a subroutine call to the relative address on the stack into the dictionary. This is not identica
as Forth's standard COMPILE which takes its argument from the input stream.

CONSTANT while compiling: (n --) while executing: (-- n) <standard> Creates a word from the next token in the input stream
which, when executed, returns the value, n.

COUNT (addr -- addr+1 length) <standard> Assuming that the relative address on the stack is the start of string data, and the
found at addr is the strings length, the address of the start of the string characters and the string length are left on the stack

CR (--) say: "sea-are" <standard> "Cr" advances the cursor to the next line. Do not confuse this word with "{cr}".

CREATE while compiling: (--) while executing: (-- addr) <standard> Create builds a word from the next token in the input strea
When the new word is executed, it returns the relative address of the cell following the words entry.

CSTATE (-- addr) say: "sea-state" Returns the relative address of a byte which is zero if the interpreter is not in 'compiling' mod
and 128 if it is.

D+ (n1 n2 n3 n4 -- n1+n3 n2+n4) say: "dee-plus" <standard> Adds the top two double numbers and leaves the double numbe
sum on the stack.

D. (d --) say: "dee-dot" <standard> Print the dvalue on the stack according to the current number base.

DABS (dval -- |dval|) say: "dabs" <standard> Return the absolute value of the dvalue on the top of the stack.

DECIMAL (--) <standard> Sets the current number base to ten.

DL! (n1 n2 daddr32 --) say: "dee-el-store" Store a 32 bit value at an absolute 32 bit address.

DL@ (daddr32 -- n1 n2) say: "dee-el-at" Get the 32 bit value from the 32 bit absolute address on the stack.

DLITERAL compiling: (d --) executing: (-- d) <standard> "Dliteral" compiles a double number from the stack. When a word
containing a dliteral is executed, the number is pushed to the stack.

DNEGATE (d -- -d) say: "dee-negate" <standard> Negate the 32 bit value on the stack.

DO (--) <standard> Compile the word "paren-do" to begin an indexed loop. Used with "loop" or "+loop". "Do" is an immediate
word and is used within a colon definition. (see "paren-do", "loop" and "+loop")

DOES> while compiling: (--) while executing: (addr --) say: "does" <standard> "Does>" is used in the definition of a defining
word, following "create" to define the run-time behavior of the defined word.

DROP (n1 n2 -- n1) <standard> Remove the value on the top of the stack.

DUP (n -- n n) say: "dupe" <standard> Duplicate the value on the top of the stack.

ELSE while compiling: (addr -- addr) while executing: (--) <standard> Used optionally between "if" and "then" in a conditional
forward branch. "Else" is an immediate word and is used within a colon definition.

EMIT (c --) <standard> Print the ASCII character represented by the value of a number on the stack.

EXECUTE (addr --) <standard> "Execute" causes the routine whose relative address is on the stack to happen.

EXIT (--) <standard> "Exit" drops an absolute address from the return stack and executes a machine language return instructio
terminating the current routine.

EXPECT (addr count --) <standard> Waits for 'count' number of characters to be typed, storing the characters, in order, at the relative address, addr. If more characters are typed, they are echoed to the screen, but not saved. A blank character and a zero byte are appended to the characters. Events are handled normally during "expect".

FILL (addr count char --) <standard> Places 'count' characters of 'char' at the relative address, 'addr'.

FORGET (--) <standard> Searches the dictionary for the next token from the input stream. If found, that word and all subsequ words are removed from the dictionary.

GROW (n --) Change the amount of reserved memory above the dictionary by 'n' bytes. If 'n' is negative the free space is redu

HEADER (--) "Header" builds a name and link field for a new word at "here".

HERE (-- addr) <standard> "Here" is the relative address of the start of free memory.

HEX (--) <standard> "Hex" sets the current number base to sixteen.

HOLD (c --) <standard> Insert the character on the stack into the number being converted to a string. Use between "greater-th sharp" and "sharp-less-than".

IF (flag --) <standard> Used with "else" and "then" to branch conditionally. A true flag causes the words following "if" and befor "then" (or "else") to be executed. "If" is an immediate word and is used within a colon definition.

ID. (addr --) say: "eye-dee-dot" <standard> Print the name of the word whose address is on the stack. Undefined characters ar represented by an ellipsis.

IMMEDIATE (--) <standard> "Immediate" is used after a definition, to flag the word as 'immediate' so that is it will execute withi colon definition, rather than being compiled into the definition. "Immediate" sets bit seven of the name length byte of the la word defined.

KEY (-- n) <standard> Waits for a character to be typed, echoes the character to the screen, advances the cursor and returns t ASCII code on the stack. Events are handled during the wait.

L! (n daddr32 --) say: "el-store" Store the value of n at the absolute address on the stack.

L@ (daddr32 -- n) say: "lat" or "el-at" Retrieve the value found at the absolute address on the stack.

LATEST (-- name.addr) <standard> Return the name address of the last word defined. Latest is used by "search" to find the e of the dictionary.

LEAVE (--) <standard> Causes a premature exit from a "do", "loop" (or "+loop") construct by setting the loop index equal to the loop limit. Use only within a definite loop structure.

LITERAL compiling: (n --) executing: (-- n) <standard> "Literal" compiles a number from the stack to the dictionary. When a word containing a literal is executed, "literal" pushes the number onto the stack.

LOOP compiling: (addr --) executing: (--) <standard> "Loop" is an immediate word used to terminate a "do" … "loop" construction, in a colon definition. Branch to the beginning of the loop if the index is less than the limit. (see "+loop" and "d

M/MOD (numer32 denom16 -- rem16 quot32) say: "em-slash-mod" <standard> Divide a double number by a single number leaving a single remainder and a double quotient.

MACRO (--) A word is flagged as a macro definition by following the word's definition with the word "macro". Macro definitions compile their code field inline rather than compiling a subroutine call. "Macro" sets bit six of the name length byte of the las word defined. (see "mcompile" and "immediate")

MAX (n1 n2 -- n) <standard> Returns the larger of the two top numbers on the stack.

MCOMPILE (addr --) "Mcompile" writes the code field found at the address on the stack. An RTS instruction signals the end of t routine and compilation stops. "Mcompile" compiles any word's code field.

MIN (n1 n2 -- n) <standard> Returns the smaller of the top two numbers on the stack.

MOD (n1 n2 -- remainder) say: "mod" <standard> Returns the remainder (but not the quotient) of 'n1' divided by 'n2'.

MON (--) Causes a monitor such as TMON (with EUA) or MacsBug to activate via the _Debugger trap. Execution will continue upon exit from the monitor.

NEGATE (n -- -n) <standard> Leave the result of zero minus n on the stack.

NULL (--) This is a no operation word.

NUMBER (addr -- n t OR f) <standard> "Number" attempts to convert the string at addr to a value according to the current number base. If the conversion is successful (that is, if all characters are numerals) the value and a true flag are left on the stack. Failure leaves only a false flag on the stack. Unlike FORTH's NUMBER, "number" does not convert double length numbers.

OPEN (--) "Open" displays the standard file dialog that allows you to select a file to be interpreted. Unlike "-->", this does not req the path to be specified. (see "load next file")

OR (n1 n2 -- n1ORn2) <standard> Leave the result of n1 OR n1 on the stack. The value is computed bitwise.

OVER (n1 n2 -- n1 n2 n1) <standard> "Over" duplicates the second number on the stack to the top of the stack.

PAD (-- addr) <standard> Leave the address of a scratch pad for numeric conversion. The pad is used downward in memory. The address of "pad" is 40 bytes beyond "here".

PAGE (--) <standard> "Page" is from the days of mechanical Teletype terminals but is still used. It clears the window and mov the cursor to the upper left hand corner.

PMODE (mode --) say: "pea-mode" Set the drawing transfer mode of the pen.

QUIT (--) <standard> "Quit" stops executing and returns to the input loop with no message. (see "abort")

R (-- n) say: "are" <standard> "Are" puts the top 16 bit number of the return stack onto the (parameter) stack. The return stack unaffected. During the execution of a definite loop ("do" … "loop") the index is kept on the top of the return stack. "Are" is used to retrieve the value of the index within these loops. (see "to-are", "are-from", "do", "loop" and "plus-loop")

R0@ (-- dabs.addr) say: "are-naught-at" Return the absolute address of the bottom of the return stack.

R> (-- n) return stack: (n --) say: "are-from" <standard> "Are-from" gets a number off the return stack and puts it on the parameter stack. (see "to-are" and "are")

REPEAT while compiling: (addr1 addr2 --) while executing: (--) <standard> "Repeat" is an immediate word, used within a col definition to terminate a "begin" …"while", … "repeat" indefinite loop. At runtime "repeat" branches unconditionally to the w following "begin" (at addr1). (see "begin" and "while")

ROOM (-- bytes) "Room" leaves the bytes of headroom above "here" on the stack. Addresses beyond the headroom should not written to, even if they are addressable, because they may be used by the system. If more room is needed, use "grow". (s "grow")

ROT (n1 n2 n3 -- n2 n3 n1) say: "rote" <standard> "Rot" brings the third stack item to the top of the stack.

RP@ (-- dabs.addr) say: "are-pea-at" Return the absolute address of the top of the return stack.

S0@ (-- dabs.addr) say: "ess-naught-at" Return the absolute address of the bottom of the parameter stack.

S>D (n -- d) say: "ess-to-dee" Make a single number on the stack into a double number using sign extension.

SAVE (--) "Save" writes the dictionary to the disk. All pertinent data, such as headroom size, the values of variables, etc. are al saved.

SEARCH (addr -- addr t OR f) "Search" looks for the next token from the input stream in the dictionary, starting with the word whose name address is on the stack. If found, its address and a true flag (minus one) are returned. If the search fails, a fa flag (zero) is left on the stack.

SIGN (n d -- d) <standard> If the single number is negative, place a negative sign into the conversion "pad". Use between "greater-than-sharp" and "sharp-less-than".

SP@ (-- dabs.addr) say: "ess-pea-at" <standard> Return the absolute address of the top of the parameter stack before the addr is put on it.

SPACE (--) <standard> "Space" prints a space character.

SWAP (n1 n2 -- n2 n1) <standard> "Swap" exchanges the top two numbers on the 8/29/24stack.

TASK (--) "Task" is a no operation word which marks the end of the dictionary. The added part of the dictionary can be removed executing: FORGET TASK   : TASK ;

THEN while compiling: (addr --) while executing: (--) <standard> "Then" is an immediate word which terminates an "if", ("else" "then" construction within a colon definition. (see "if" and "then")

TIB (-- addr) say: "tib" (rhymes with rib) <standard> "Tib" returns the relative address of the terminal input buffer. The buffer is a 84 byte data area below the dictionary. The input stream usually points to a byte within "tib".

TOKEN (--) "Token" moves the next word from the input stream to "here", the end of the dictionary.

TYPE (addr n --) <standard> "Type" prints n number of characters from memory starting at the relative address, addr. For best results addr should contain at least n ASCII characters.

U. (n --) say: "you-dot" <standard> Print the value of n in the current numeric base as an unsigned number.

U* (n1 n2 -- d[n1*n2]) say: "you-star" <standard> "You-star" multiplies two unsigned 16 bit numbers from the stack and leaves double number product on the stack.

UNTIL (flag --) <standard> "Unitl" is an immediate word used with in a colon definition after "begin" to conditionally terminate a indefinite loop. If flag is true execution passes to the next word. If false it branches back to the word following "begin". (s "begin" and "back")

UPPER (addr --) Given a string's address on the stack, "upper" converts lower case to upper case. The first byte must contain th length of the string.

VARIABLE compiling: (--) executing: (-- addr) <standard> "Variable" creates a word from the next word in the input stream, an reserves one cell (two bytes) of data. When the new word is executed, it leaves the relative address of the data cell on the stack. Words created with "variable" are four byte macros.

WHAZAT (--) say: "what-is-that" "Whazat" prints the current token from the input stream, and executes "abort", if the current tok was not found in the dictionary.

WHILE (flag --) <standard> "While" is an immediate word used within a colon definition, between "begin" and "repeat" to contro indefinite loop with an exit in the middle. If the flag is true, the words after "while" and before "repeat" are executed, if the f is false, execution jumps to the word following "repeat". (see "begin" and "repeat".)

WORD (c --) <standard> "Word" moves the next token, delimited by the character on the stack, from the input stream to "here
the end of the dictionary.

WORDS (--) <standard> "Words" prints all of the words in the dictionary.

XOR (n1 n2 --) <standard> say: "zor" or "eks-or" Leave the result of a bitwise exclusive or of n1 and n2 on the stack.

[(--) say: "left-bracket" <standard> "Left-bracket" sets the interpreter into immediate mode. Words following "left-bracket" are
executed rather than compiled. "Left-bracket" is an immediate word.

[COMPILE] (--) say: "bracket-compile" <standard> "Bracket-compile" is an immediate word used within a colon definition to
compile the following immediate word from the input stream into the current definition.

] (--) say: "right-bracket" <standard> "Right-bracket" puts the interpreter into compile mode so that subsequent non-immediate
words will be compiled to the dictionary. "Right-bracket" is an immediate word.

"{cr}" (--) say: "" When the token consisting of an ascii 13 is executed, the return stack is reset, and the input sequence is resta
This is an immediate word. Do not confuse this word with "cr". In the DA version, "{cr}" causes input to be taken from the f
stack.

"{null}" (--) say: "" <standard> This is an alias for "{cr}" to assure that "expect"ed text, which is terminated by a null (zero byte), a
pasted text is treated the same. In the DA version, "{null}" specifies that the next input is to be taken from the keyboard. T
is an immediate word.