



PnP Script

The complete language guide.

Write source. Understand the language. Build locally.

104 reference entries with syntax, explanations and original examples. A searchable, linked reference you can keep offline.

READ / UNDERSTAND / BUILD

Use Studio to compile PnP Script into inject.bin. Keep the editable .txt source alongside the binary and select the keyboard layout used by the receiving computer.

Studio never runs a payload or accesses USB hardware. Finished firmware and physical device validation are pending.

STUDIO 1.0.0 / PRE-RELEASE DOCUMENTATION

studio.plugnpwn.io/learn/

FIND YOUR WAY

One language. Full reference.

Start with the essentials, then jump to the construct you need. Category titles and page numbers are clickable. The PDF bookmark panel lists every entry; the quick index at the end links directly to each one.

Before you build	3
Setup, examples, boundaries and how to read this guide.	
Start here	5
Source files, keyboard layouts, comments and the build process.	
Text & keys	8
Literal text, key combinations, held keys and raw keyboard reports.	
Values & expressions	16
Global variables, constants and explicit expression grouping.	
Control flow	21
Branches, loops, functions and payload lifecycle.	
Compile-time tools	25
Macros, conditional compilation, repetition and reusable source.	
Device & USB	34
USB personas, descriptors, physical-button actions and status LEDs.	
Randomness & timing	40
Delays, random values, random keystrokes and jitter.	
Host & storage	44
Lock-key feedback, storage waits and on-card data.	
Reserved variables	49
The complete device-variable namespace and its access rules.	
Keep your source	72
Saving editable files and attribution.	
Source example license	73
MIT permission, copyright and notices.	
Quick index	74
Every entry, alphabetically, with a direct page link.	

START HERE

Before you build.

Write. Build. Keep your source.

The language guide for PlugnPwn Studio. Write source locally, build inject.bin, and keep your editable .txt source alongside the binary.

Studio 1.0.0 is in pre-release development. This guide documents the supported source language and planned device contract; it does not certify finished firmware or hardware behavior.

Start with a blank document

Examples that type text assume a blank editor on your own computer. Build in Studio and save the source. The planned payload-transfer workflow copies inject.bin to the root of the installed microSD card through the device USB storage mode; safely eject, then unplug and reconnect to run it. See "A first payload" for button behavior and its overrides. Finished firmware and device validation are pending.

Firmware flashing uses a separate volume: hold the button before connecting USB and copy the firmware .uf2 file to the bootloader volume, with or without a microSD card. Do not copy inject.bin there. Direct card-reader preparation is optional when the card is accessible, such as before assembly. Studio does not execute your payload, flash firmware, or access USB hardware.

A successful build verifies supported source syntax and binary encoding. Physical behavior still depends on firmware, USB enumeration, the receiving application and its keyboard layout. The final device validation is a separate release step.

[READ THE REFERENCE](#)

Reading this guide

Syntax shows the grammar; placeholders such as `expression` or `key` are not literal source.

Complete example includes its declarations and function definitions. It can be built independently.

Fragment identifies a call site that needs additional source; it is never presented as a complete payload.

Build time means Studio resolves the construct before exporting bytes. Device runtime means the firmware performs it when the payload runs.

Persistent card write means the operation changes files on microSD.

Use uppercase command/key names, case-sensitive names and one instruction per line. Variables begin with `$`; the `$_` namespace belongs to the device. Comments use `REM` or `REM_BLOCK / END_REM`. A saved UTF-8 text file is the editable source; `inject.bin` is its compiled output.

Source whitespace can be significant. Example panels preserve indentation and trailing spaces, but a PDF viewer may normalize whitespace when copying. Use Copy or Save example in the online guide when you need an exact source file.

Resource bounds

Studio applies source, nesting, expansion, diagnostic and binary-size limits. These are implementation limits that keep compilation bounded, not promises about device execution time. A runtime loop or event wait can outlast any displayed count of static delays. The current numeric limits and byte-format decisions are maintained in the compiler contract.

01 / LANGUAGE REFERENCE

Start here

Source files, keyboard layouts, comments and the build process.

001 / BUILD TIME AND DEVICE RUNTIME

A first payload

Write text in Studio and build a binary for the planned device workflow.

Effect: USB/host interaction when run on the device.

SYNTAX

```
One instruction per line
```

PnP Script is the source language. Studio compiles it locally into inject.bin; the browser does not execute the payload or connect to USB hardware.

Save the editable source as a UTF-8 .txt file. For payload transfer, the planned firmware exposes the installed microSD card as a USB storage volume. Copy inject.bin to its root, safely eject it, then unplug and reconnect to run the new payload. Finished firmware and device validation are pending.

The planned default button press after normal boot stops a payload and exposes storage. A payload waiting for the button, a custom BUTTON_DEF handler or DISABLE_BUTTON can override that action. It does not prevent an existing payload from starting before the press. A missing or malformed inject.bin is planned to enter storage automatically.

Firmware flashing is separate: hold the button before connecting USB, then copy the firmware .uf2 file to the bootloader volume. This does not require a microSD card and is not where inject.bin belongs. A card reader is an optional preparation route only when the card is accessible, such as before assembly.

Payload execution belongs to the device firmware. A successful build checks the source and encoding; it does not certify behavior on a particular operating system or device.

Commands and named keys are uppercase. Source names and variable/function references are case-sensitive. Blank lines separate ideas without emitting instructions.

COMPLETE EXAMPLE

```
REM Open a blank text editor on your own computer first.
ATTACKMODE HID
DELAY 1500
STRINGLN Hello from PlugnPwn.
```

002 / BUILD TIME

Keyboard layout

Compile for the keyboard layout configured on the receiving computer.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
Select the host keyboard layout before building.
```

Text is converted to USB keyboard usages when the payload is built. The host keyboard layout determines which characters those usages produce.

Studio includes standard PC keyboard maps for US English, UK English, German and French. Select the map matching the receiving host, then rebuild after any change. Custom layouts and macOS-specific variants may differ.

Random-character operations also need the selected layout. The compiler includes a compact runtime key table when a payload needs one.

Some punctuation uses a dead key followed by SPACE to produce the spacing character. German ^ and grave accent, and applicable French accents, can therefore require two physical presses. Use STRING for composed characters; shortcuts accept a single physical key.

Unmappable characters produce a diagnostic. UTF-8 source permits readable comments without implying that every Unicode character can be typed by every keyboard layout.

COMPLETE EXAMPLE

```
STRING Az09!  
ENTER
```

003 / BUILD TIME

REM · REM_BLOCK · END_REM

Explain your payload without adding device instructions.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
REM text
REM_BLOCK
    comment text
END_REM
```

REM comments run to the end of their line. REM_BLOCK and END_REM enclose a multiline comment.

Comment text is removed at compile time. A comment-only source has no executable output.

Close every block explicitly. Use REM for comments; do not assume # or // starts a comment.

COMPLETE EXAMPLE

```
REM A greeting for a blank document.
REM_BLOCK
    This note can span several lines.
    Ελληνικά and other Unicode are fine in comments.
END_REM
STRINGLN Welcome.
```

02 / LANGUAGE REFERENCE

Text & keys

Literal text, key combinations, held keys and raw keyboard reports.

004 / DEVICE RUNTIME

STRING

Type literal text without pressing Enter.

Effect: USB/host interaction when run on the device.

SYNTAX

```
STRING text
```

One separator after STRING is syntax; the following text is data. Additional leading spaces and trailing spaces can be intentional keystrokes.

Text from consecutive STRING instructions is joined on the receiving computer unless a key instruction separates it.

Runtime variables are not interpolated into text: STRING \$COUNT types the characters \$COUNT. A DEFINE macro may still replace matching source text at compile time.

COMPLETE EXAMPLE

```
STRING Hello,  
STRING world!
```

Preserve trailing whitespace on source line 1 (1 SPACE).

005 / DEVICE RUNTIME

STRINGLN

Type literal text and then press Enter.

Effect: USB/host interaction when run on the device.

SYNTAX

```
STRINGLN text
```

STRINGLN adds an Enter keystroke after its text. It does not insert a platform-specific byte string such as CRLF into a file.

The selected keyboard layout applies to every literal character. Preserve spaces you intend to type.

COMPLETE EXAMPLE

```
STRINGLN First line.
STRINGLN Second line.
```

006 / DEVICE RUNTIME

STRING blocks · STRINGLN blocks

Write multiline text without repeating the opener.

Effect: USB/host interaction when run on the device.

SYNTAX

```
STRING
    text
END_STRING
STRINGLN
    text
END_STRINGLN
```

A bare STRING or STRINGLN starts a text block. Close STRING with END_STRING and STRINGLN with END_STRINGLN. A mismatched terminator is an error.

Lines inside the block are text, not commands. A line containing DELAY 100 types that text.

STRING concatenates its body lines and strips their leading formatting whitespace. STRINGLN adds Enter after each body line and removes the block formatting indentation while retaining additional indentation.

Use four spaces for each formatting level and inspect the result when indentation matters. Empty lines in STRINGLN produce an Enter.

An inline terminator may close a same-line string, for example STRING Hello END_STRING. The inline terminator must match the opener; mismatched names produce a diagnostic.

STRING blocks · STRINGLN blocks / continued

COMPLETE EXAMPLE

```
STRINGLN
  First line.
  Second line.
END_STRINGLN
```

007 / DEVICE RUNTIME

STRING_<LANG> · STRINGLN_<LANG>

Label the language of a block of text being typed.

Effect: USB/host interaction when run on the device.

SYNTAX

```
STRING_<LANG>
  text
END_STRING
STRINGLN_<LANG>
  text
END_STRINGLN
```

Available suffixes: POWERSHELL, BASH, BATCH, PYTHON, HTML, RUBY and JAVASCRIPT.

These labels describe text. Studio does not run the embedded language, validate its program or contact an interpreter.

Use a bare suffixed block opener, inline text, or an explicit same-line terminator. The suffix labels the text language and does not change how inline text is typed.

Available command pairs: STRING_POWERSHELL / STRINGLN_POWERSHELL, STRING_BASH / STRINGLN_BASH, STRING_BATCH / STRINGLN_BATCH, STRING_PYTHON / STRINGLN_PYTHON, STRING_HTML / STRINGLN_HTML, STRING_RUBY / STRINGLN_RUBY, STRING_JAVASCRIPT / STRINGLN_JAVASCRIPT.

STRING_PYTHON retains indentation using the indented-block rule even though it does not append Enter. Other STRING suffixes follow ordinary STRING block whitespace behavior.

COMPLETE EXAMPLE

```
STRINGLN_HTML
  <p>Hello from PlugnPwn.</p>
END_STRINGLN
```

008 / DEVICE RUNTIME

Named keys & aliases

Send navigation, editing, system or function keys.

Effect: USB/host interaction when run on the device.

SYNTAX

```
ENTER · ESCAPE / ESC · BACKSPACE · TAB · SPACE
UPARROW / UP · DOWNARROW / DOWN · LEFTARROW / LEFT · RIGHTARROW / RIGHT
HOME · END · PAGEUP · PAGEDOWN · INSERT · DELETE / DEL
PAUSE / BREAK · PRINTSCREEN · MENU / APP
F1 · F2 · F3 · F4 · F5 · F6 · F7 · F8 · F9 · F10 · F11 · F12
CAPSLOCK · NUMLOCK · SCROLLLOCK
```

Each ordinary key instruction presses and releases that key. A plain printable character on its own line also works; STRING is clearer for text.

F0 is a compatibility alias for F10, and SCROLLOCK is an alias for SCROLLLOCK. Prefer the conventional spellings in new source.

Lock keys change host keyboard state. Host applications decide what navigation, function and system keys do.

COMPLETE EXAMPLE

```
STRING First
TAB
STRING Second
ENTER
```

009 / DEVICE RUNTIME

Modifier combinations

Press a key together with explicit modifier keys.

Effect: USB/host interaction when run on the device.

SYNTAX

```
CTRL / CONTROL / LEFTCTRL / LEFTCONTROL · SHIFT / LEFTSHIFT
ALT / OPTION / LEFTALT · GUI / WINDOWS / COMMAND / LEFTGUI
RIGHTCTRL / RIGHTCONTROL · RIGHTSHIFT · RIGHTALT / ALTGR · RIGHTGUI
CTRL SHIFT a
CTRL-SHIFT a
COMMAND-OPTION-SHIFT a
```

Write modifiers before one ordinary key, separated by spaces or hyphens. Aliases map to the same physical modifier bit.

Each combination is a complete press/release pair. Modifier state does not carry into the next command unless HOLD is used.

Spell SHIFT explicitly in shortcuts and use an unshifted key name, for example CTRL-SHIFT a. A character that normally needs Shift is not a reliable shorthand for a shortcut.

Modifier combinations / continued

A modifier on its own is handled by INJECT_MOD. ALT-TAB is the named Alt+Tab combination; avoid it when you want the TAB key alone.

COMPLETE EXAMPLE

```
REM Use a blank document on your own computer.
STRING Sample
CTRL a
```

010 / DEVICE RUNTIME

INJECT_MOD

Press, hold or release a modifier without an ordinary key.

Effect: USB/host interaction when run on the device.

SYNTAX

```
INJECT_MOD SHIFT
INJECT_MOD
SHIFT
INJECT_MOD
HOLD SHIFT
INJECT_MOD
RELEASE SHIFT
```

INJECT_MOD marks a standalone modifier operation. The next relevant keyboard action may be a press, HOLD or RELEASE.

A bare INJECT_MOD in the active source enables standalone modifier names across branches and functions. Studio reasserts the flag before a standalone press when needed, so its meaning stays a key even if an earlier flag-setting branch does not run.

Prefer an immediately adjacent modifier operation so the scope is obvious. Explicitly release every held modifier.

The one-line press form accepts a modifier name or a hyphenated combination, such as INJECT_MOD CTRL-ALT.

COMPLETE EXAMPLE

```
INJECT_MOD
HOLD SHIFT
STRING a
INJECT_MOD
RELEASE SHIFT
RESET
```

011 / DEVICE RUNTIME

HOLD · RELEASE · RESET

Keep a key down across instructions and release it deliberately.

Effect: USB/host interaction when run on the device.

SYNTAX

```
HOLD key
RELEASE key
RESET
```

HOLD adds a key to the held set; RELEASE removes the named key. Other held keys remain down.

The host controls auto-repeat timing. A sustained key-down is not equivalent to a counted number of key presses.

RESET clears pending keyboard work and releases all held keys and modifiers. It does not restart the payload.

The boot-keyboard report has six ordinary key slots plus modifier bits. Stay within that limit; simultaneous combinations need device testing.

COMPLETE EXAMPLE

```
HOLD a
DELAY 100
RELEASE a
RESET
```

012 / DEVICE RUNTIME

INJECT · KEYCODE

Send an explicit two-byte keyboard unit.

Effect: USB/host interaction when run on the device.

SYNTAX

```
INJECT 0400
KEYCODE 0x0402
```

Write four hexadecimal digits in file order: two digits for the USB keyboard usage, then two for the modifier mask. 0400 is the A key without Shift; 0402 includes Left Shift.

The 0x prefix is optional. Prefer the explicit four-digit form instead of depending on padding.

Raw values bypass character-to-layout mapping. They are keyboard reports, not ASCII codes or text encodings.

COMPLETE EXAMPLE

```
INJECT 0400
KEYCODE 0x0402
```

013 / DEVICE RUNTIME

KEY_DOWN / UP · MOD_DOWN / UP · MOD_KEY_DOWN / UP

Control the key and modifier portions of the held report.

Effect: USB/host interaction when run on the device.

SYNTAX

```
KEY_DOWN 0400 · KEY_UP 0400
MOD_DOWN 0002 · MOD_UP 0002
MOD_KEY_DOWN 0402 · MOD_KEY_UP 0402
```

KEY_DOWN and KEY_UP affect the usage byte; MOD_DOWN and MOD_UP affect the modifier byte. MOD_KEY_DOWN and MOD_KEY_UP apply both.

All these operations share the same held state as HOLD and RELEASE. Finish with matching releases or RESET.

Operands use the same usage-first hexadecimal spelling as INJECT. Raw reports still depend on the host interpreting USB keyboard usages.

COMPLETE EXAMPLE

```
KEY_DOWN 0400
DELAY 100
KEY_UP 0400
MOD_DOWN 0002
MOD_UP 0002
MOD_KEY_DOWN 0402
MOD_KEY_UP 0402
RESET
```

014 / DEVICE RUNTIME

INJECT_VAR

Type the keyboard unit stored in a numeric variable.

Effect: USB/host interaction when run on the device.

SYNTAX

```
INJECT_VAR $KEY
```

A numeric keyboard unit has its HID usage in the low byte and modifiers in the high byte. Thus numeric 0x0204 means usage 04 with modifier 02.

This differs from raw source spelling INJECT 0402, which writes bytes in file order.

Storing a random keycode in a variable draws once; repeatedly injecting that variable repeats the same selected character. It does not print the decimal value of a variable.

COMPLETE EXAMPLE

```
VAR $KEY = 0x0204
INJECT_VAR $KEY
```

Values & expressions

Global variables, constants and explicit expression grouping.

VAR - assignment

Store unsigned 16-bit values in global payload variables.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
VAR $COUNT = 3
$COUNT = ( $COUNT - 1 )
VAR $COUNT = ( $COUNT + 1 )
```

Values are unsigned integers from 0 through 65535. Decimal and 0x-prefixed hexadecimal literals are accepted.

Variables are global to the payload, including inside functions. There are no function parameters or per-call local variables.

An assignment can introduce a variable; repeating VAR on an existing name assigns a new value. Top-level reads need a preceding declaration or assignment. Function and button-handler bodies can reference globals assigned later in the source; initialize shared values before calling the function or activating the handler.

Use names such as \$COUNT and \$READY. The \$_ prefix belongs to reserved device variables.

Arithmetic wraps modulo 65536 in the PnP runtime contract. Prefer ranges that avoid relying on overflow.

COMPLETE EXAMPLE

```
VAR $COUNT = 3
$COUNT = ( $COUNT - 1 )
IF ( $COUNT == 2 ) THEN
    STRINGLN Two.
END_IF
```

016 / DEVICE RUNTIME

TRUE · FALSE

Use boolean constants and explicit comparisons.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
TRUE
FALSE
```

FALSE is zero; TRUE is one. At runtime, zero is false and a nonzero condition is true.

Logical and comparison expressions produce a boolean result. Conditional-compilation directives test macro tokens differently: use literal TRUE or FALSE there.

COMPLETE EXAMPLE

```
VAR $READY = TRUE
IF $READY THEN
    STRINGLN Ready.
END_IF
```

017 / DEVICE RUNTIME

Arithmetic operators

Calculate numeric values with explicit grouping.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
+ · - · * · / · % · ^
```

^ means exponentiation, not bitwise XOR. / is integer division and % is the remainder operation.

Each grouping level contains one binary operation. Write (2 + (3 * 4)) rather than relying on implicit mathematical precedence.

Avoid zero divisors and out-of-range intermediate assumptions. Runtime numeric behavior belongs to the device interpreter; Studio encodes the expression.

COMPLETE EXAMPLE

```
VAR $AREA = ( 7 * 6 )
VAR $POWER = ( 2 ^ 3 )
VAR $REMAINDER = ( 17 % 5 )
IF ( $AREA == 42 ) THEN
    STRINGLN Calculation complete.
END_IF
```

018 / DEVICE RUNTIME

Comparison operators

Compare two numeric values.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
== . != . > . < . >= . <=
```

Comparisons return TRUE or FALSE. Use == for equality; = performs assignment.

Compare named OS constants directly with \$_OS instead of hard-coding numeric IDs.

COMPLETE EXAMPLE

```
VAR $LEVEL = 4
IF ( $LEVEL >= 3 ) THEN
    STRINGLN Level reached.
ELSE
    STRINGLN Keep going.
END_IF
```

019 / DEVICE RUNTIME

Logical operators

Combine boolean conditions.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
&& / AND
|| / OR
```

&& (AND) requires both values to be true; || (OR) requires at least one. The word aliases must be uppercase.

Group each comparison and the surrounding operation. Do not rely on short-circuit evaluation to suppress side effects or protect an invalid calculation.

COMPLETE EXAMPLE

```
VAR $A = 3
VAR $B = 7
IF ( ( $A > 0 ) && ( $B < 10 ) ) THEN
    STRINGLN Both conditions hold.
END_IF
```

020 / DEVICE RUNTIME

Bitwise operators

Work with masks and shifted integer values.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
& · | · << · >>
```

& and | combine individual bits. << and >> shift an unsigned 16-bit value.

Use bounded shift counts and explicit parentheses. ^ is power and is not an XOR operator.

COMPLETE EXAMPLE

```
VAR $FLAGS = 3
VAR $LOW_BIT = ( $FLAGS & 1 )
VAR $SHIFTED = ( $LOW_BIT << 2 )
IF ( $SHIFTED == 4 ) THEN
    STRINGLN Bit set.
END_IF
```

021 / BUILD TIME

Expression grouping

Make evaluation order unambiguous.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
VAR $VALUE = ( 4 * 10 ) + 2
VAR $VALUE = ( ( 4 * 10 ) + 2 )
```

Parentheses define the expression tree. The outermost pair may be omitted when each inner operation is already grouped.

A lone literal, variable or no-argument function call is also an expression. More than one ungrouped operator at a level is rejected.

Parse nesting is bounded so malformed or excessively deep expressions cannot grow without limit.

COMPLETE EXAMPLE

```
VAR $VALUE = ( 4 * 10 ) + 2
IF ( $VALUE == 42 ) THEN
    STRINGLN Forty-two.
END_IF
```

022 / DEVICE RUNTIME

Operating-system constants

Name an OS classification produced by your own detection code.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

WINDOWS · MACOS · LINUX · CHROMEOS · ANDROID · IOS

These constants are values in expressions; they are not standalone commands.

\$_OS is a writable classification slot. It is not automatically populated by a browser feature or a magic firmware detector.

Include and review any source function that sets \$_OS. Host classification is heuristic and should not be treated as proof of platform identity.

COMPLETE EXAMPLE

```
VAR $EXPECTED_OS = LINUX
IF ( $_OS == $EXPECTED_OS ) THEN
    STRINGLN Matching classification.
END_IF
```

04 / LANGUAGE REFERENCE

Control flow

Branches, loops, functions and payload lifecycle.

023 / DEVICE RUNTIME

IF · ELSE IF · ELSE · END_IF

Choose one branch from a conditional chain.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
IF ( condition ) THEN
    instructions
ELSE IF ( condition ) THEN
    instructions
ELSE
    instructions
END_IF
IF $FLAG THEN
```

The first true branch executes. ELSE is optional and runs only when earlier conditions are false.

One END_IF closes the whole chain. Nested conditions each need their own terminator.

A single variable condition may omit parentheses: IF \$READY THEN.

COMPLETE EXAMPLE

```
VAR $COUNT = 2
IF ( $COUNT == 1 ) THEN
    STRINGLN One.
ELSE IF ( $COUNT == 2 ) THEN
    STRINGLN Two.
ELSE
    STRINGLN Another value.
END_IF
```

024 / DEVICE RUNTIME

WHILE · END_WHILE

Repeat a block while its condition remains true.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
WHILE ( condition )
    instructions
END_WHILE
WHILE $FLAG
WHILE TRUE
```

The condition is checked before each iteration. Update the loop variable yourself; WHILE does not increment or decrement it.

WHILE TRUE intentionally keeps running until payload control or a device action stops it. Prefer a bounded loop for a first test.

Loops execute on the device. Studio compiles the body once; it does not run the loop to build the binary.

COMPLETE EXAMPLE

```
VAR $COUNT = 3
WHILE ( $COUNT > 0 )
    STRINGLN Tick.
    $COUNT = ( $COUNT - 1 )
END_WHILE
```

025 / DEVICE RUNTIME

FUNCTION · END_FUNCTION

Group reusable instructions behind a no-argument call.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
FUNCTION Greet()
    instructions
END_FUNCTION
Greet()
```

A function definition is skipped during ordinary sequential execution. Its body runs when the function is called.

Functions take no arguments. Share data through global variables and use a return value when needed. Do not call a shared helper from a button handler while the same helper is active in the main program.

Function calls may appear as statements or in expressions. PnP Script v1 rejects direct and indirect recursion because variables and compiler temporary registers are shared rather than saved per call.

Acyclic nested calls are supported.

FUNCTION · END_FUNCTION / continued

COMPLETE EXAMPLE

```
FUNCTION Greet()
    STRINGLN Hello again.
    RETURN 0
END_FUNCTION
Greet()
Greet()
```

026 / DEVICE RUNTIME

RETURN

Leave a function and supply its numeric result.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
RETURN expression
RETURN
```

RETURN exits the current function immediately. Write an explicit return value on every path when a caller uses the result.

Bare RETURN returns zero. Reaching END_FUNCTION without RETURN also returns zero. RETURN belongs inside a function, not at top level. Prefer an explicit value in reusable functions.

Function values use the payload runtime return mechanism; do not assume local variables or per-call persistent result objects.

COMPLETE EXAMPLE

```
FUNCTION Answer()
    RETURN 42
END_FUNCTION
VAR $ANSWER = Answer()
IF ( $ANSWER == 42 ) THEN
    STRINGLN Answer received.
END_IF
```

027 / DEVICE RUNTIME

STOP_PAYLOAD · RESTART_PAYLOAD

End the current run or restart the resident program.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
STOP_PAYLOAD  
RESTART_PAYLOAD
```

STOP_PAYLOAD stops the main instruction stream. It does not erase the payload file or act as RESET for held keys.

RESTART_PAYLOAD returns to the beginning of the image already held in device memory; it does not reload a newly copied file from the card.

Button-handler availability is a separate state. Disable or retire a handler explicitly if it should no longer run.

A restart can reapply USB descriptors and trigger enumeration when resolved settings change.

COMPLETE EXAMPLE

```
STRINGLN Finished.  
STOP_PAYLOAD
```

05 / LANGUAGE REFERENCE

Compile-time tools

Macros, conditional compilation, repetition and reusable source.

028 / BUILD TIME

DEFINE

Substitute a source token before compilation.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
DEFINE NAME replacement text
DEFINE #NAME replacement text
```

DEFINE is textual substitution, not a runtime variable. Its value can be a number, boolean or text with spaces.

Ordinary macro names replace whole whitespace-delimited tokens in text and complete identifier tokens in expressions, including next to parentheses and operators. A # prefixed macro can substitute inside a larger text token.

Text that does not match a macro remains text. Runtime \$variables inside STRING are not automatically expanded.

Macro expansion is bounded and recursive cycles are diagnosed. Prefer distinct descriptive names rather than command keywords.

COMPLETE EXAMPLE

```
DEFINE MESSAGE Hello from a macro.
DEFINE #NAME PlugnPwn
STRINGLN MESSAGE
STRINGLN Device: #NAME
```

029 / BUILD TIME

IF_DEFINED_TRUE · IF_NOT_DEFINED_TRUE

Choose source branches before any bytes are emitted.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
IF_DEFINED_TRUE #FLAG
    source
ELSE_DEFINED
    source
END_IF_DEFINED
IF_NOT_DEFINED_TRUE #FLAG
```

IF_DEFINED_TRUE selects its first arm only for a macro whose value is the exact token TRUE.

IF_NOT_DEFINED_TRUE selects its first arm for FALSE or an undefined macro. Numeric 1 and 0 are not substitutes for TRUE and FALSE in this directive family.

ELSE_DEFINED changes the active arm; END_IF_DEFINED closes the block. Nesting is allowed and every enclosing arm must be active for a line to survive.

Disabled source emits no instructions. Unclosed and misplaced directive boundaries are errors, not permission to silently discard the rest of a file.

COMPLETE EXAMPLE

```
DEFINE #GREETING TRUE
IF_DEFINED_TRUE #GREETING
    STRINGLN Hello.
ELSE_DEFINED
    STRINGLN Goodbye.
END_IF_DEFINED
```

030 / BUILD TIME

REPEAT

Expand a fixed number of copies at build time.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
REPEAT count command
REPEAT count
```

The inline form emits count copies of the following instruction. It does not create a runtime loop.

The bare form repeats the previous repeatable statement count additional times. Prefer the inline form when sharing source because its target is explicit.

Expansion and final binary size are bounded. For a large or data-dependent count, use a bounded WHILE loop instead.

COMPLETE EXAMPLE

```
REPEAT 3 STRINGLN Again.
```

031 / BUILD TIME

STAGE · END_STAGE

Organize source into named sections.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
STAGE name
    instructions
END_STAGE
```

Stages are source labels. They do not delay, jump, restart or save runtime state.

The instructions inside execute in normal sequence. Names exist to make a longer source easier to read.

COMPLETE EXAMPLE

```
STAGE Greeting
    STRINGLN First stage.
END_STAGE
STAGE Closing
    STRINGLN Second stage.
END_STAGE
```

032 / BUILD TIME

EXTENSION · VERSION · END_EXTENSION

Keep reusable source together with a local version label.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
EXTENSION Name
VERSION 1.0
    source
END_EXTENSION
```

An extension is included source, not a downloaded plug-in or an additional binary format.

Functions inside follow normal definition/call rules. Ordinary top-level instructions inside the wrapper still execute in sequence.

VERSION is local source metadata. Studio does not fetch a remote version, execute a dependency installer or automatically resolve missing extension bodies.

COMPLETE EXAMPLE

```
EXTENSION FriendlyGreeting
VERSION 1.0
FUNCTION Greet()
    STRINGLN Hello from this extension.
    RETURN 0
END_FUNCTION
END_EXTENSION
Greet()
```

033 / BUILD TIME

OS_DETECT · TRANSLATE source libraries

Include implementation source before calling a library function.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
DETECT_OS()
TRANSLATE_INT()
TRANSLATE_HEX()
TRANSLATE_BOOL()
```

These names describe conventional source libraries, not built-in device opcodes. The example is a call-site fragment and needs the corresponding FUNCTION definitions.

DETECTION code assigns \$_OS using available device observations. It is a heuristic whose behavior must be tested against the host systems you support.

TRANSLATE functions read \$INPUT and type its numeric or boolean value as text. STRING \$INPUT instead types the literal token.

OS_DETECT · TRANSLATE source libraries / continued

Use locally reviewed source. Studio performs no network import and supplies no secret host-execution environment.

FRAGMENT - INCLUDE THE REQUIRED DEFINITIONS FIRST

```
REM Include the reviewed library source above these calls.
DETECT_OS()
VAR $INPUT = 42
TRANSLATE_INT()
```

034 / BUILD TIME

DEBUGGER_BREAKPOINT · INJECT_BREAKPOINT_LINE_NUMBER

Add visible, device-side checkpoints to your own test payload.

Effect: USB/host interaction when run on the device.

SYNTAX

```
DEBUGGER_BREAKPOINT
INJECT_BREAKPOINT_LINE_NUMBER
```

DEBUGGER_BREAKPOINT emits a visible checkpoint label with the original source line and waits for the physical button. It is not a browser debugger or a host-process breakpoint.

INJECT_BREAKPOINT_LINE_NUMBER types the source-line number followed by a semicolon, then waits 800 ms. DEBUGGER_BREAKPOINT places its label between Enter keystrokes and changes the LED from red while waiting to green afterward. These are PnP Script source-level expansions.

Use a blank document on your own computer so the marker text has an obvious destination.

COMPLETE EXAMPLE

```
STRINGLN Before the checkpoint.
DEBUGGER_BREAKPOINT
STRINGLN After the button press.
```

035 / BUILD TIME AND DEVICE RUNTIME

TRANSLATE_INT: complete source

Type a numeric value as decimal text using a locally included function.

Effect: USB/host interaction when run on the device.

SYNTAX

```
VAR $INPUT = 42
TRANSLATE_INT()
```

TRANSLATE_INT: complete source / continued

This original, complete example defines every function it calls. It prints 42; changing \$INPUT to zero prints 0.

The function handles the full unsigned 16-bit range, removes leading zeroes and leaves \$INPUT unchanged.

PnP_PrintDigit uses ordinary STRING instructions, so the selected keyboard layout applies instead of assuming US HID digit positions.

The \$PNP_ variables are global scratch storage. Reserve those names for the helper and do not call the helper recursively or from an interrupting handler while it is already running.

COMPLETE EXAMPLE

```
VAR $INPUT = 42
VAR $PNP_DIGIT = 0
FUNCTION PnP_PrintDigit()
  IF ( $PNP_DIGIT == 0 ) THEN
    STRING 0
  ELSE IF ( $PNP_DIGIT == 1 ) THEN
    STRING 1
  ELSE IF ( $PNP_DIGIT == 2 ) THEN
    STRING 2
  ELSE IF ( $PNP_DIGIT == 3 ) THEN
    STRING 3
  ELSE IF ( $PNP_DIGIT == 4 ) THEN
    STRING 4
  ELSE IF ( $PNP_DIGIT == 5 ) THEN
    STRING 5
  ELSE IF ( $PNP_DIGIT == 6 ) THEN
    STRING 6
  ELSE IF ( $PNP_DIGIT == 7 ) THEN
    STRING 7
  ELSE IF ( $PNP_DIGIT == 8 ) THEN
    STRING 8
  ELSE IF ( $PNP_DIGIT == 9 ) THEN
    STRING 9
  END_IF
  RETURN 0
END_FUNCTION
FUNCTION TRANSLATE_INT()
  VAR $PNP_DEC_VALUE = $INPUT
  VAR $PNP_DEC_PLACE = 10000
  VAR $PNP_DEC_STARTED = FALSE
  WHILE ( $PNP_DEC_PLACE > 0 )
    VAR $PNP_DIGIT = ( $PNP_DEC_VALUE / $PNP_DEC_PLACE )
    IF ( ( $PNP_DIGIT > 0 ) || ( $PNP_DEC_STARTED || ( $PNP_DEC_PLACE == 1 ) ) ) THEN
      PnP_PrintDigit()
      $PNP_DEC_STARTED = TRUE
    END_IF
    $PNP_DEC_VALUE = ( $PNP_DEC_VALUE % $PNP_DEC_PLACE )
    $PNP_DEC_PLACE = ( $PNP_DEC_PLACE / 10 )
  END_WHILE
  RETURN 0
```

TRANSLATE_INT: complete source / continued

COMPLETE EXAMPLE / CONTINUED

```
END_FUNCTION
TRANSLATE_INT()
ENTER
```

036 / BUILD TIME AND DEVICE RUNTIME

TRANSLATE_HEX: complete source

Type a numeric value as four uppercase hexadecimal digits.

Effect: USB/host interaction when run on the device.

SYNTAX

```
VAR $INPUT = 42
TRANSLATE_HEX()
```

This original, complete helper prints 002A for 42. Its format is four uppercase hexadecimal digits with no 0x prefix.

Add STRING 0x before the call if that prefix is useful. \$INPUT is preserved.

The helper uses ordinary layout-aware text instructions, not numeric ASCII-to-HID assumptions.

Reserve its \$PNP_ scratch names and avoid re-entering it from a button handler. When combining the decimal and hex helpers, include PnP_PrintDigit only once.

COMPLETE EXAMPLE

```
VAR $INPUT = 42
VAR $PNP_DIGIT = 0
FUNCTION PnP_PrintDigit()
IF ( $PNP_DIGIT == 0 ) THEN
    STRING 0
ELSE IF ( $PNP_DIGIT == 1 ) THEN
    STRING 1
ELSE IF ( $PNP_DIGIT == 2 ) THEN
    STRING 2
ELSE IF ( $PNP_DIGIT == 3 ) THEN
    STRING 3
ELSE IF ( $PNP_DIGIT == 4 ) THEN
    STRING 4
ELSE IF ( $PNP_DIGIT == 5 ) THEN
    STRING 5
ELSE IF ( $PNP_DIGIT == 6 ) THEN
    STRING 6
ELSE IF ( $PNP_DIGIT == 7 ) THEN
    STRING 7
ELSE IF ( $PNP_DIGIT == 8 ) THEN
    STRING 8
ELSE IF ( $PNP_DIGIT == 9 ) THEN
    STRING 9
END_IF
RETURN 0
```

TRANSLATE_HEX: complete source / continued

COMPLETE EXAMPLE / CONTINUED

```

END_FUNCTION
FUNCTION TRANSLATE_HEX()
VAR $PNP_HEX_VALUE = $INPUT
VAR $PNP_HEX_PLACES = 4
WHILE ( $PNP_HEX_PLACES > 0 )
    $PNP_HEX_PLACES = ( $PNP_HEX_PLACES - 1 )
    VAR $PNP_DIGIT = ( ( $PNP_HEX_VALUE >> ( $PNP_HEX_PLACES * 4 ) ) & 15 )
    IF ( $PNP_DIGIT < 10 ) THEN
        PnP_PrintDigit()
    ELSE IF ( $PNP_DIGIT == 10 ) THEN
        STRING A
    ELSE IF ( $PNP_DIGIT == 11 ) THEN
        STRING B
    ELSE IF ( $PNP_DIGIT == 12 ) THEN
        STRING C
    ELSE IF ( $PNP_DIGIT == 13 ) THEN
        STRING D
    ELSE IF ( $PNP_DIGIT == 14 ) THEN
        STRING E
    ELSE
        STRING F
    END_IF
END_WHILE
RETURN 0
END_FUNCTION
TRANSLATE_HEX()
ENTER

```

037 / BUILD TIME AND DEVICE RUNTIME

TRANSLATE_BOOL: complete source

Type TRUE or FALSE from a numeric value.

Effect: USB/host interaction when run on the device.

SYNTAX

```
VAR $INPUT = TRUE
TRANSLATE_BOOL()
```

The complete helper prints FALSE for zero and TRUE for any nonzero input.

The body is ordinary PnP Script source. There is no remote dependency or special browser/runtime translation opcode.

Copy the function definition with its call site; a call alone is not a complete library.

COMPLETE EXAMPLE

```
VAR $INPUT = TRUE
FUNCTION TRANSLATE_BOOL()
IF $INPUT THEN
    STRING TRUE
ELSE
    STRING FALSE
END_IF
RETURN 0
END_FUNCTION
TRANSLATE_BOOL()
ENTER
```

06 / LANGUAGE REFERENCE

Device & USB

USB personas, descriptors, physical-button actions and status LEDs.

038 / DEVICE RUNTIME

ATTACKMODE

Choose which USB functions the host can see.

Effect: USB/host interaction when run on the device.

SYNTAX

```
ATTACKMODE HID
ATTACKMODE STORAGE
ATTACKMODE HID STORAGE
ATTACKMODE OFF
```

HID exposes the keyboard interface. STORAGE exposes the microSD-backed mass-storage interface. HID STORAGE combines them.

OFF logically disconnects the USB device while the payload can continue running. It is not a power switch.

Changing a mode or a resolved descriptor can disconnect and re-enumerate the device. Give the receiving computer time to recognize it.

The mode is required when ATTACKMODE is present. Prefer declaring it explicitly at the start of a payload.

COMPLETE EXAMPLE

```
ATTACKMODE HID
DELAY 1500
STRINGLN Keyboard mode.
```

039 / DEVICE RUNTIME

USB descriptor parameters

Set the identifiers and strings for a USB persona.

Effect: USB/host interaction when run on the device.

SYNTAX

```
ATTACKMODE HID VID_1234 PID_5678
ATTACKMODE HID MAN_PlugnPwn PROD_LabDevice SERIAL_1234
VID_$VENDOR · PID_$PRODUCT
```

USB descriptor parameters / continued

VID_ and PID_ are a paired set of four-digit hexadecimal vendor and product identifiers. Use identifiers you are entitled to use for your own hardware.

MAN_, PROD_ and SERIAL_ form a second all-or-nothing set. Manufacturer and product are 1-32 alphanumeric characters; serial is 1-12 decimal digits.

Parameter groups may appear in any order. Prefer explicit complete pairs/sets so the intended persona is clear.

Raw variable-backed VID/PID values use the protocol register byte order. Prefer literal hex descriptors; consult the binary contract before generating identifiers from variables.

OFF does not apply descriptor parameters. Keep its source line simply ATTACKMODE OFF.

COMPLETE EXAMPLE

```
ATTACKMODE HID MAN_PlugnPwn PROD_LabDevice SERIAL_1234
DELAY 1500
STRINGLN Descriptor test.
```

040 / DEVICE RUNTIME

Random descriptor parameters

Resolve fresh identifiers when the device executes a persona change.

Effect: USB/host interaction when run on the device.

SYNTAX

```
VID_RANDOM · PID_RANDOM
MAN_RANDOM · PROD_RANDOM · SERIAL_RANDOM
```

Random descriptor values are generated on the device at runtime, not by Studio during compilation.

MAN_RANDOM and PROD_RANDOM generate twelve uppercase letters; SERIAL_RANDOM generates twelve digits.

The pairing rules still apply: supply VID and PID together, and all three descriptor-string fields together.

Changing descriptors can make the host recognize a different device instance. Use this only in your own device/host test setup.

COMPLETE EXAMPLE

```
ATTACKMODE HID MAN_RANDOM PROD_RANDOM SERIAL_RANDOM
DELAY 1500
STRINGLN Random descriptor test.
```

041 / DEVICE RUNTIME

SAVE_ATTACKMODE · RESTORE_ATTACKMODE

Temporarily change the USB persona and restore its complete settings.

Effect: USB/host interaction when run on the device.

SYNTAX

```
SAVE_ATTACKMODE
RESTORE_ATTACKMODE
```

The saved slot includes mode, numeric IDs and descriptor strings. A later save replaces that slot.

Restoring can re-enumerate the device if the settings differ. It is not a saved payload or card backup.

COMPLETE EXAMPLE

```
ATTACKMODE HID
SAVE_ATTACKMODE
ATTACKMODE OFF
DELAY 1000
RESTORE_ATTACKMODE
DELAY 1500
STRINGLN Back in keyboard mode.
```

042 / DEVICE RUNTIME

WAIT_FOR_BUTTON_PRESS

Pause the instruction stream until the device button is pressed.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
WAIT_FOR_BUTTON_PRESS
```

The firmware continues servicing USB while the payload waits. Studio does not simulate the physical button.

A direct button wait has priority over a user button handler and remains meaningful even while handler/default-button actions are disabled.

This is a runtime wait with no compile-time duration estimate.

COMPLETE EXAMPLE

```
ATTACKMODE HID
LED_R
WAIT_FOR_BUTTON_PRESS
LED_G
STRINGLN Button received.
```

043 / DEVICE RUNTIME

BUTTON_DEF · END_BUTTON

Define what a later button press should do.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
BUTTON_DEF
    instructions
END_BUTTON
```

Reaching the definition registers the handler and skips its body. A subsequent button press runs it according to firmware button routing.

A handler may interrupt a delay and then return to the interrupted instruction stream. Keep it short and deliberate.

\$_BUTTON_TIMEOUT controls the cooldown between counted presses. Set \$_BUTTON_USER_DEFINED to FALSE to retire the handler.

Defining a handler is distinct from WAIT_FOR_BUTTON_PRESS. A direct wait consumes its press before the handler is considered.

COMPLETE EXAMPLE

```
BUTTON_DEF
    LED_G
END_BUTTON
LED_R
DELAY 3000
STRINGLN Main sequence finished.
```

044 / DEVICE RUNTIME

ENABLE_BUTTON · DISABLE_BUTTON

Enable or disable button handler/default actions.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
ENABLE_BUTTON  
DISABLE_BUTTON
```

These commands update `$_BUTTON_ENABLED`. While disabled, ordinary handler/default actions are not invoked.

A pending `WAIT_FOR_BUTTON_PRESS` still resumes from a press. The `BOOTSEL` power-on flashing path is separate from these runtime commands.

COMPLETE EXAMPLE

```
DISABLE_BUTTON  
DELAY 500  
ENABLE_BUTTON  
LED_G
```

045 / DEVICE RUNTIME

LED_R · LED_G · LED_OFF

Set the red, green or off state of the device indicator.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
LED_R / LED_RED  
LED_G / LED_GREEN  
LED_OFF
```

The aliases `LED_RED` and `LED_GREEN` encode the corresponding short form.

Firmware system/activity indicators may take precedence. Use the indicator-control variables when you need to distinguish payload feedback from automatic status.

COMPLETE EXAMPLE

```
LED_R  
DELAY 500  
LED_G  
DELAY 500  
LED_OFF
```

046 / DEVICE RUNTIME

ENABLE_SYSTEM_LEDS · DISABLE_SYSTEM_LEDS

Control automatic system indication separately from direct LED commands.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
ENABLE_SYSTEM_LEDS  
DISABLE_SYSTEM_LEDS
```

These are the command forms of writing TRUE or FALSE to `$_SYSTEM_LEDS_ENABLED`.

Storage, injection and capture indication have separate reserved controls. Do not assume that one switch suppresses every possible indicator.

COMPLETE EXAMPLE

```
DISABLE_SYSTEM_LEDS  
LED_G  
DELAY 500  
ENABLE_SYSTEM_LEDS
```

07 / LANGUAGE REFERENCE

Randomness & timing

Delays, random values, random keystrokes and jitter.

047 / DEVICE RUNTIME

DELAY

Wait for a literal or runtime number of milliseconds.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
DELAY 1000
DELAY $WAIT
```

A decimal literal is resolved at build time and encoded as delay units. A variable-backed delay reads its unsigned 16-bit value when the device executes it.

Literal delays accept nonnegative integer milliseconds within compiler and output-size bounds. Do not write fractions or negative waits.

The build summary counts static delays in encoded source; it cannot predict runtime loops, variable delays, USB enumeration or event waits.

Choose a delay from tests of the receiving application, not from a promise that every host is ready after a fixed duration.

COMPLETE EXAMPLE

```
VAR $WAIT = 250
DELAY $WAIT
STRINGLN After a short wait.
```

048 / DEVICE RUNTIME

RANDOM_* character commands

Type one runtime-selected character from a defined class.

Effect: USB/host interaction when run on the device.

SYNTAX

```
RANDOM_LOWERCASE_LETTER · RANDOM_UPPERCASE_LETTER
RANDOM_LETTER · RANDOM_NUMBER · RANDOM_SPECIAL
RANDOM_CHAR / RANDOM_CHARACTER
```

Letters use a-z and A-Z; RANDOM_NUMBER uses 0-9. RANDOM_SPECIAL uses !@#\$%^&*(). RANDOM_CHAR is their union.

RANDOM_* character commands / continued

The random draw happens on the device. Studio embeds the selected layout table when these operations need it.

These values are for payload behavior and test data, not a claim of cryptographically secure password or key generation.

COMPLETE EXAMPLE

```
STRING Sample:
RANDOM_UPPERCASE_LETTER
RANDOM_LOWERCASE_LETTER
RANDOM_NUMBER
ENTER
```

Preserve trailing whitespace on source line 1 (1 SPACE).

049 / DEVICE RUNTIME

Runtime random numbers

Choose a bounded integer or retain one sampled value.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_MIN = 0
$_RANDOM_MAX = 9
VAR $SAMPLE = $_RANDOM_INT
```

\$_RANDOM_INT is re-evaluated whenever it is referenced. Copy it into a normal variable to keep one draw stable.

Keep the minimum at or below the maximum and both within 0-65535.

Random keyboard-unit variables and random ASCII-code variables are different: one represents a HID unit; the other represents a numeric character value.

COMPLETE EXAMPLE

```
$_RANDOM_MIN = 1
$_RANDOM_MAX = 6
VAR $SAMPLE = $_RANDOM_INT
IF ( $SAMPLE >= 1 ) THEN
    STRINGLN Sample stored.
END_IF
```

050 / DEVICE RUNTIME

Inter-keystroke jitter

Vary the device-side gap between typed keys.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
$_JITTER_ENABLED = TRUE
$_JITTER_MAX = 20
```

Jitter affects the gap after release and before the next key-down; it does not change the meaning of DELAY.

The runtime draws a gap from zero through the configured maximum in milliseconds. The design defaults are disabled and a 20 ms maximum.

PRNG/seed behavior belongs to the device runtime. Studio cannot promise that separate physical runs produce identical timings.

COMPLETE EXAMPLE

```
$_JITTER_MAX = 20
$_JITTER_ENABLED = TRUE
STRINGLN Timing sample.
$_JITTER_ENABLED = FALSE
```

051 / BUILD TIME AND DEVICE RUNTIME

STRING_DELAY

Type literal text with a fixed pause between its characters.

Effect: USB/host interaction when run on the device.

SYNTAX

```
STRING_DELAY milliseconds text
```

The compiler places a delay between successive source characters, with no initial or trailing pause. It does not pause between the dead-key strokes composing one character. No Enter is added automatically; DEFAULT_DELAY can append its separate statement-level pause.

Milliseconds is an unsigned decimal or hexadecimal literal. Text follows the same selected keyboard map as STRING; a composed character may emit more than one physical keystroke.

This expands at build time and increases binary size. Runtime variables are literal text here, just as they are in STRING.

COMPLETE EXAMPLE

```
STRING_DELAY 40 Hello.
ENTER
```

052 / BUILD TIME AND DEVICE RUNTIME

DEFAULT_DELAY · DEFAULTDELAY

Set a build-time delay appended to subsequent executable statements.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
DEFAULT_DELAY milliseconds
DEFAULTDELAY milliseconds
DEFAULT_DELAY 0
```

Both spellings have the same meaning. Use a nonnegative literal in milliseconds; zero disables the default.

This is a source-order compiler setting. It affects emitted text, assignments and ordinary commands, including an explicit DELAY; it is not a runtime variable and does not add a delay to structural block boundaries or RETURN.

Instructions inside branches and functions inherit the setting active when their source is compiled. Prefer explicit DELAY instructions when timing across reusable functions or conditional branches must be obvious.

The compiler does not run conditions when applying this setting. Put configuration at top level and reset it explicitly before unrelated source.

COMPLETE EXAMPLE

```
DEFAULT_DELAY 100
STRINGLN First.
STRINGLN Second.
DEFAULT_DELAY 0
STRINGLN Done.
```

08 / LANGUAGE REFERENCE

Host & storage

Lock-key feedback, storage waits and on-card data.

053 / DEVICE RUNTIME

WAIT_FOR_* lock-state commands

Wait for a host-reported lock state or state change.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
WAIT_FOR_CAPS_ON · WAIT_FOR_CAPS_OFF · WAIT_FOR_CAPS_CHANGE  
WAIT_FOR_NUM_ON · WAIT_FOR_NUM_OFF · WAIT_FOR_NUM_CHANGE  
WAIT_FOR_SCROLL_ON · WAIT_FOR_SCROLL_OFF · WAIT_FOR_SCROLL_CHANGE
```

ON and OFF wait for the corresponding state; CHANGE waits for a transition. The device observes reports from the host.

Host lock reporting varies. `$_RECEIVED_HOST_LOCK_LED_REPLY` indicates whether a report has been observed; no Studio build can guarantee that a host will send one.

Event waits can remain blocked when the expected external event never occurs. Keep your test workflow able to recover using the device button or a power cycle.

COMPLETE EXAMPLE

```
STRINGLN Toggle Caps Lock to continue.  
WAIT_FOR_CAPS_CHANGE  
STRINGLN Lock-state change received.
```

054 / DEVICE RUNTIME

SAVE_HOST_KEYBOARD_LOCK_STATE · RESTORE_HOST_KEYBOARD_LOCK_STATE

Snapshot lock states and restore them with keyboard actions.

Effect: USB/host interaction when run on the device.

SYNTAX

```
SAVE_HOST_KEYBOARD_LOCK_STATE
RESTORE_HOST_KEYBOARD_LOCK_STATE
```

The saved snapshot covers Caps Lock, Num Lock and Scroll Lock. Read it through the corresponding `$_SAVED_*_ON` variables.

Restore compares current and saved state and sends the needed lock-key presses. Reliable behavior requires host feedback.

This affects the host keyboard state; it is separate from a USB persona snapshot.

COMPLETE EXAMPLE

```
SAVE_HOST_KEYBOARD_LOCK_STATE
CAPSLOCK
DELAY 250
RESTORE_HOST_KEYBOARD_LOCK_STATE
```

055 / DEVICE RUNTIME

WAIT_FOR_STORAGE_ACTIVITY · WAIT_FOR_STORAGE_INACTIVITY

Coordinate a payload with USB mass-storage activity.

Effect: Runtime device state or an external-event wait.

SYNTAX

```
WAIT_FOR_STORAGE_ACTIVITY
WAIT_FOR_STORAGE_INACTIVITY
```

Use a storage-capable persona first. Activity means host storage commands, including polls that may transfer no file data.

Inactivity waits for the configured quiet period. Some hosts poll continuously, so this is advisory timing, not proof that a file was safely copied or flushed.

`$_STORAGE_ACTIVITY_TIMEOUT` is a live countdown reloaded on activity. Writes set the reload interval; reads may show the remaining time.

The device writing its own log or restoring a payload is not host USB-storage activity.

WAIT_FOR_STORAGE_ACTIVITY · WAIT_FOR_STORAGE_INACTIVITY / continued

COMPLETE EXAMPLE

```
ATTACKMODE STORAGE
$_STORAGE_ACTIVITY_TIMEOUT = 1000
WAIT_FOR_STORAGE_INACTIVITY
LED_G
```

056 / DEVICE RUNTIME

EXFIL

Append a payload variable to the device data file.

Effect: Persistent card write when run on the device.

SYNTAX

```
EXFIL $VALUE
```

This example records only a constant created by the payload. It does not read a host file.

The PnP runtime contract appends the variable as two little-endian bytes to loot.bin on microSD. Existing data is retained.

EXFIL is independent of lock-state capture and does not require \$_EXFIL_MODE_ENABLED.

This is a persistent card write. Review or remove the resulting file using your normal device/card workflow.

COMPLETE EXAMPLE

```
VAR $SAMPLE = 0x1234
EXFIL $SAMPLE
LED_G
```

057 / DEVICE RUNTIME

Lock-state capture

Enable a device-side receiver for an explicitly arranged test stream.

Effect: Persistent card write when run on the device.

SYNTAX

```
$_EXFIL_MODE_ENABLED = TRUE / FALSE
```

This flag enables the runtime lock-report capture path. It does not itself read data from the host or create a host-side sender.

The protocol uses Caps Lock and Num Lock transitions as data bits and Scroll Lock as a completion signal. Turning the flag off flushes/closes captured data in loot.bin.

Use only with a test stream you deliberately arrange on your own host. The example leaves capture disabled.

Captured bytes persist on the card. The compiler does not inspect, upload or interpret that file.

COMPLETE EXAMPLE

```
$_EXFIL_MODE_ENABLED = FALSE
LED_G
```

058 / DEVICE RUNTIME

HIDE_PAYLOAD · RESTORE_PAYLOAD

Remove or rewrite the running payload files on microSD.

Effect: Persistent card write when run on the device.

SYNTAX

```
HIDE_PAYLOAD
RESTORE_PAYLOAD
```

HIDE_PAYLOAD removes inject.bin and the seed file if present; it is not a hidden-file attribute. The running image remains in device memory.

RESTORE_PAYLOAD rewrites the resident image and seed to the card, including the runtime variable values defined by the firmware contract.

Perform these operations while the card is not exposed over USB mass storage: use OFF or HID.

Unplugging after removal without restoring leaves no payload file to load next time. Keep a separate source and binary copy before testing file lifecycle behavior.

COMPLETE EXAMPLE

```
ATTACKMODE OFF
HIDE_PAYLOAD
RESTORE_PAYLOAD
```

HIDE_PAYLOAD · RESTORE_PAYLOAD / continued

COMPLETE EXAMPLE / CONTINUED

```
ATTACKMODE HID
DELAY 1500
STRINGLN Payload files restored.
```

059 / DEVICE RUNTIME

Inspect host observations

Use device observations without assuming an unverified OS classification.

Effect: USB/host interaction when run on the device.

SYNTAX

```
$_HOST_CONFIGURATION_REQUEST_COUNT
$_RECEIVED_HOST_LOCK_LED_REPLY
```

This is a complete observation example, not a claimed operating-system detector. It checks whether a lock report has actually arrived.

USB request count and lock-report behavior are inputs a locally included classifier can inspect. They are not unique proof of all six OS values.

Keep \$_OS at zero until your included classification code has a reason to assign a value. Validate a classifier against your actual firmware and supported hosts before depending on it.

COMPLETE EXAMPLE

```
ATTACKMODE HID
DELAY 1500
VAR $REQUESTS = $_HOST_CONFIGURATION_REQUEST_COUNT
IF $_RECEIVED_HOST_LOCK_LED_REPLY THEN
    STRINGLN Host lock feedback is available.
ELSE
    STRINGLN No lock feedback has been observed.
END_IF
```

09 / LANGUAGE REFERENCE

Reserved variables

The complete device-variable namespace and its access rules.

060 / DEVICE RUNTIME

\$_BUTTON_ENABLED

Enable handler/default button actions; direct waits have priority.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_BUTTON_ENABLED
```

Access: read/write. Enable handler/default button actions; direct waits have priority.

Design default: TRUE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_BUTTON_ENABLED
LED_G
```

061 / DEVICE RUNTIME

\$_BUTTON_USER_DEFINED

A reached button definition is active. Clear it to retire the handler; do not set it without a defined handler.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_BUTTON_USER_DEFINED
```

Access: read/write. A reached button definition is active. Clear it to retire the handler; do not set it without a defined handler.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_BUTTON_USER_DEFINED
LED_G
```

062 / DEVICE RUNTIME

\$_BUTTON_PUSH_RECEIVED

Records whether a button press has been received.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_BUTTON_PUSH_RECEIVED
```

Access: read/write. Records whether a button press has been received.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_BUTTON_PUSH_RECEIVED
LED_G
```

063 / DEVICE RUNTIME

\$_BUTTON_TIMEOUT

Cooldown in milliseconds between counted presses; distinct from electrical debounce.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_BUTTON_TIMEOUT
```

Access: read/write. Cooldown in milliseconds between counted presses; distinct from electrical debounce.

Design default: 1000. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_BUTTON_TIMEOUT
LED_G
```

064 / DEVICE RUNTIME

\$_SYSTEM_LEDS_ENABLED

Enable automatic boot and persona-change indication.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_SYSTEM_LEDS_ENABLED
```

Access: read/write. Enable automatic boot and persona-change indication.

Design default: TRUE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_SYSTEM_LEDS_ENABLED
LED_G
```

065 / DEVICE RUNTIME

\$_STORAGE_LEDS_ENABLED

Enable indication associated with USB storage reads/writes.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_STORAGE_LEDS_ENABLED
```

Access: read/write. Enable indication associated with USB storage reads/writes.

Design default: TRUE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_STORAGE_LEDS_ENABLED
LED_G
```

066 / DEVICE RUNTIME

\$LED_SHOW_STORAGE_ACTIVITY

Enable the continuous storage active/idle indicator.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$LED_SHOW_STORAGE_ACTIVITY
```

Access: read/write. Enable the continuous storage active/idle indicator.

Design default: TRUE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $LED_SHOW_STORAGE_ACTIVITY
LED_G
```

067 / DEVICE RUNTIME

\$INJECTING_LEDS_ENABLED

Enable indication while keyboard work is being sent.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$INJECTING_LEDS_ENABLED
```

Access: read/write. Enable indication while keyboard work is being sent.

Design default: TRUE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $INJECTING_LEDS_ENABLED
LED_G
```

068 / DEVICE RUNTIME

\$_EXFIL_LEDS_ENABLED

Enable indication while lock-state capture is active.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_EXFIL_LEDS_ENABLED
```

Access: read/write. Enable indication while lock-state capture is active.

Design default: TRUE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_EXFIL_LEDS_ENABLED
LED_G
```

069 / DEVICE RUNTIME

\$_CAPSLOCK_ON

Current Caps Lock state reported by the host.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_CAPSLOCK_ON
```

Access: read-only. Current Caps Lock state reported by the host.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_CAPSLOCK_ON
LED_G
```

070 / DEVICE RUNTIME

\$_NUMLOCK_ON

Current Num Lock state reported by the host.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_NUMLOCK_ON
```

Access: read-only. Current Num Lock state reported by the host.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_NUMLOCK_ON
LED_G
```

071 / DEVICE RUNTIME

\$_SCROLLLOCK_ON

Current Scroll Lock state reported by the host.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_SCROLLLOCK_ON
```

Access: read-only. Current Scroll Lock state reported by the host.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_SCROLLLOCK_ON
LED_G
```

072 / DEVICE RUNTIME

\$_SAVED_CAPSLOCK_ON

Saved Caps Lock state from attachment or an explicit lock-state snapshot.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_SAVED_CAPSLOCK_ON
```

Access: read-only. Saved Caps Lock state from attachment or an explicit lock-state snapshot.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_SAVED_CAPSLOCK_ON
LED_G
```

073 / DEVICE RUNTIME

\$_SAVED_NUMLOCK_ON

Saved Num Lock state from attachment or an explicit lock-state snapshot.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_SAVED_NUMLOCK_ON
```

Access: read-only. Saved Num Lock state from attachment or an explicit lock-state snapshot.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_SAVED_NUMLOCK_ON
LED_G
```

074 / DEVICE RUNTIME

\$_SAVED_SCROLLLOCK_ON

Saved Scroll Lock state from attachment or an explicit lock-state snapshot.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_SAVED_SCROLLLOCK_ON
```

Access: read-only. Saved Scroll Lock state from attachment or an explicit lock-state snapshot.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_SAVED_SCROLLLOCK_ON
LED_G
```

075 / DEVICE RUNTIME

\$_RECEIVED_HOST_LOCK_LED_REPLY

Whether a host lock-state report has been received.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RECEIVED_HOST_LOCK_LED_REPLY
```

Access: read-only. Whether a host lock-state report has been received.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RECEIVED_HOST_LOCK_LED_REPLY
LED_G
```

076 / DEVICE RUNTIME

\$_EXFIL_MODE_ENABLED

Enable/disable lock-state capture to the card; disabling flushes and closes captured data.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_EXFIL_MODE_ENABLED
```

Access: read/write. Enable/disable lock-state capture to the card; disabling flushes and closes captured data.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_EXFIL_MODE_ENABLED  
LED_G
```

077 / DEVICE RUNTIME

\$_STORAGE_ACTIVITY_TIMEOUT

Live inactivity countdown in milliseconds; a write sets the interval reloaded by later host activity.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_STORAGE_ACTIVITY_TIMEOUT
```

Access: read/write. Live inactivity countdown in milliseconds; a write sets the interval reloaded by later host activity.

Design default: 1000. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_STORAGE_ACTIVITY_TIMEOUT  
LED_G
```

078 / DEVICE RUNTIME

\$_PAYLOAD_PARSE_SPEED

Reserved parser-speed setting. The current firmware design initializes it to 2; further timing effects are not established.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_PAYLOAD_PARSE_SPEED
```

Access: read/write. Reserved parser-speed setting. The current firmware design initializes it to 2; further timing effects are not established.

Design default: 2. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_PAYLOAD_PARSE_SPEED
LED_G
```

079 / DEVICE RUNTIME

\$_CURRENT_VID

Current USB vendor identifier in the protocol register byte order, which is byte-swapped relative to the descriptor literal.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_CURRENT_VID
```

Access: read-only. Current USB vendor identifier in the protocol register byte order, which is byte-swapped relative to the descriptor literal.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_CURRENT_VID
LED_G
```

080 / DEVICE RUNTIME

\$_CURRENT_PID

Current USB product identifier in the protocol register byte order, which is byte-swapped relative to the descriptor literal.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_CURRENT_PID
```

Access: read-only. Current USB product identifier in the protocol register byte order, which is byte-swapped relative to the descriptor literal.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_CURRENT_PID
LED_G
```

081 / DEVICE RUNTIME

\$_OS

OS classification assigned by included detection source: WINDOWS, MACOS, LINUX, CHROMEOS, ANDROID or IOS; zero means unclassified.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_OS
```

Access: read/write. OS classification assigned by included detection source: WINDOWS, MACOS, LINUX, CHROMEOS, ANDROID or IOS; zero means unclassified.

Design default: 0. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_OS
LED_G
```

082 / DEVICE RUNTIME

\$_HOST_CONFIGURATION_REQUEST_COUNT

Count of host configuration/descriptor requests. A write resets or sets the counter; the firmware continues counting from there.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_HOST_CONFIGURATION_REQUEST_COUNT
```

Access: read/write. Count of host configuration/descriptor requests. A write resets or sets the counter; the firmware continues counting from there.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_HOST_CONFIGURATION_REQUEST_COUNT
LED_G
```

083 / DEVICE RUNTIME

\$_CURRENT_ATTACKMODE

Current persona: 0 OFF, 1 HID, 2 STORAGE, 3 HID STORAGE.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_CURRENT_ATTACKMODE
```

Access: read-only. Current persona: 0 OFF, 1 HID, 2 STORAGE, 3 HID STORAGE.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_CURRENT_ATTACKMODE
LED_G
```

084 / DEVICE RUNTIME

\$_JITTER_ENABLED

Enable randomized gaps between keystrokes.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_JITTER_ENABLED
```

Access: read/write. Enable randomized gaps between keystrokes.

Design default: FALSE. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_JITTER_ENABLED
LED_G
```

085 / DEVICE RUNTIME

\$_JITTER_MAX

Maximum inter-keystroke jitter in milliseconds.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_JITTER_MAX
```

Access: read/write. Maximum inter-keystroke jitter in milliseconds.

Design default: 20. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_JITTER_MAX
LED_G
```

086 / DEVICE RUNTIME

\$_STORAGE_ACTIVE

Live host storage-activity flag in the PnP firmware design.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_STORAGE_ACTIVE
```

Access: read-only. Live host storage-activity flag in the PnP firmware design.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_STORAGE_ACTIVE
LED_G
```

087 / DEVICE RUNTIME

\$_RANDOM_INT

A fresh integer in the inclusive configured random minimum/maximum range on each read.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_INT
```

Access: read-only. A fresh integer in the inclusive configured random minimum/maximum range on each read.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_INT
LED_G
```

088 / DEVICE RUNTIME

\$_RANDOM_MIN

Inclusive lower bound for \$_RANDOM_INT, within 0-65535.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_MIN
```

Access: read/write. Inclusive lower bound for \$_RANDOM_INT, within 0-65535.

Design default: 0. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_MIN  
LED_G
```

089 / DEVICE RUNTIME

\$_RANDOM_MAX

Inclusive upper bound for \$_RANDOM_INT, at least the minimum and at most 65535.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_MAX
```

Access: read/write. Inclusive upper bound for \$_RANDOM_INT, at least the minimum and at most 65535.

Design default: 9. Device lifecycle and preceding payload instructions may change the current value.

This example snapshots the current value into a normal variable. It does not print or upload the value.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_MAX  
LED_G
```

090 / DEVICE RUNTIME

\$_RANDOM_SEED

Runtime PRNG seed value exposed by the firmware; the seed file is managed on the device.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_SEED
```

Access: read-only. Runtime PRNG seed value exposed by the firmware; the seed file is managed on the device.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_SEED
LED_G
```

091 / DEVICE RUNTIME

\$_RANDOM_UINT16

A fresh unsigned 16-bit random value; also used by random USB numeric descriptors.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_UINT16
```

Access: read-only. A fresh unsigned 16-bit random value; also used by random USB numeric descriptors.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_UINT16
LED_G
```

092 / DEVICE RUNTIME

\$_RANDOM_ASCII_LOWER_LETTER

A random lowercase ASCII letter code, not a HID keyboard unit.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_ASCII_LOWER_LETTER
```

Access: read-only. A random lowercase ASCII letter code, not a HID keyboard unit.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_ASCII_LOWER_LETTER  
LED_G
```

093 / DEVICE RUNTIME

\$_RANDOM_ASCII_UPPER_LETTER

A random uppercase ASCII letter code; used by random manufacturer/product strings.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_ASCII_UPPER_LETTER
```

Access: read-only. A random uppercase ASCII letter code; used by random manufacturer/product strings.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_ASCII_UPPER_LETTER  
LED_G
```

094 / DEVICE RUNTIME

\$_RANDOM_ASCII_LETTER

A random uppercase or lowercase ASCII letter code.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_ASCII_LETTER
```

Access: read-only. A random uppercase or lowercase ASCII letter code.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_ASCII_LETTER
LED_G
```

095 / DEVICE RUNTIME

\$_RANDOM_ASCII_NUMBER

A random ASCII digit code; used by random serial strings.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_ASCII_NUMBER
```

Access: read-only. A random ASCII digit code; used by random serial strings.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_ASCII_NUMBER
LED_G
```

096 / DEVICE RUNTIME

\$_RANDOM_ASCII_SPECIAL

A random ASCII code from !@#\$%^&*().

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_ASCII_SPECIAL
```

Access: read-only. A random ASCII code from !@#\$%^&*().

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_ASCII_SPECIAL
LED_G
```

097 / DEVICE RUNTIME

\$_RANDOM_ASCII_CHAR

A random ASCII letter, digit or supported special-character code.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_ASCII_CHAR
```

Access: read-only. A random ASCII letter, digit or supported special-character code.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_ASCII_CHAR
LED_G
```

098 / DEVICE RUNTIME

\$_RANDOM_LOWER_LETTER_KEYCODE

A random lowercase-letter HID keyboard unit under the compiled keyboard layout.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_LOWER_LETTER_KEYCODE
```

Access: read-only. A random lowercase-letter HID keyboard unit under the compiled keyboard layout.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

The low byte is HID usage and the high byte is modifiers. Use INJECT_VAR to type that unit; STRING does not expand numeric variables.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_LOWER_LETTER_KEYCODE  
LED_G
```

099 / DEVICE RUNTIME

\$_RANDOM_UPPER_LETTER_KEYCODE

A random uppercase-letter HID keyboard unit under the compiled keyboard layout.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_UPPER_LETTER_KEYCODE
```

Access: read-only. A random uppercase-letter HID keyboard unit under the compiled keyboard layout.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

The low byte is HID usage and the high byte is modifiers. Use INJECT_VAR to type that unit; STRING does not expand numeric variables.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_UPPER_LETTER_KEYCODE  
LED_G
```

100 / DEVICE RUNTIME

\$_RANDOM_NUMBER_KEYCODE

A random digit HID keyboard unit under the compiled keyboard layout.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_NUMBER_KEYCODE
```

Access: read-only. A random digit HID keyboard unit under the compiled keyboard layout.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

The low byte is HID usage and the high byte is modifiers. Use INJECT_VAR to type that unit; STRING does not expand numeric variables.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_NUMBER_KEYCODE  
LED_G
```

101 / DEVICE RUNTIME

\$_RANDOM_SPECIAL_KEYCODE

A random special-character HID keyboard unit under the compiled keyboard layout.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_SPECIAL_KEYCODE
```

Access: read-only. A random special-character HID keyboard unit under the compiled keyboard layout.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

The low byte is HID usage and the high byte is modifiers. Use INJECT_VAR to type that unit; STRING does not expand numeric variables.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_SPECIAL_KEYCODE  
LED_G
```

102 / DEVICE RUNTIME

\$_RANDOM_CHAR_KEYCODE

A random character HID keyboard unit under the compiled keyboard layout.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_CHAR_KEYCODE
```

Access: read-only. A random character HID keyboard unit under the compiled keyboard layout.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

The low byte is HID usage and the high byte is modifiers. Use INJECT_VAR to type that unit; STRING does not expand numeric variables.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_CHAR_KEYCODE  
LED_G
```

103 / DEVICE RUNTIME

\$_RANDOM_LETTER_KEYCODE

A random uppercase/lowercase-letter HID keyboard unit under the compiled keyboard layout.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_RANDOM_LETTER_KEYCODE
```

Access: read-only. A random uppercase/lowercase-letter HID keyboard unit under the compiled keyboard layout.

This example snapshots the current value into a normal variable. It does not print or upload the value.

Read the reserved value once into a user variable when you want to reuse a stable sample; a later direct read can draw again.

The low byte is HID usage and the high byte is modifiers. Use INJECT_VAR to type that unit; STRING does not expand numeric variables.

COMPLETE EXAMPLE

```
VAR $SNAPSHOT = $_RANDOM_LETTER_KEYCODE  
LED_G
```

104 / DEVICE RUNTIME

Reserved-variable aliases

Use alternate names without allocating unrelated user variables.

Effect: No direct host or persistent-file effect in the construct itself.

SYNTAX

```
$_LED_CONTINUOUS_SHOW_STORAGE_ACTIVITY → $_LED_SHOW_STORAGE_ACTIVITY
$_LED_SHOW_CAPS → $_CAPSLOCK_ON
$_LED_SHOW_NUM → $_NUMLOCK_ON
$_LED_SHOW_SCROLL → $_SCROLLLOCK_ON
```

The syntax list shows equivalent names, not assignments to perform. The storage-indicator alias is read/write; the three host-lock aliases are read-only.

Prefer `$_LED_SHOW_STORAGE_ACTIVITY`, `$_CAPSLOCK_ON`, `$_NUMLOCK_ON` and `$_SCROLLLOCK_ON` in new source.

Unknown `$_` names are diagnosed rather than silently becoming ordinary variables.

COMPLETE EXAMPLE

```
VAR $CAPS = $_LED_SHOW_CAPS
VAR $NUM = $_LED_SHOW_NUM
VAR $SCROLL = $_LED_SHOW_SCROLL
LED_G
```

KEEP BUILDING

Keep your source

The binary contains device instructions, not the source comments or a self-describing project archive. Keep the .txt file and note the chosen keyboard layout. Rebuild after changing either. A displayed SHA-256 describes the exported bytes; it is not inserted into the binary.

About this guide

This guide and its examples are original PlugnPwn documentation. The downloadable PDF and the online reference share the same reviewed command catalog.

Software examples use the project MIT license. PlugnPwn brand artwork and bundled fonts retain their separate notices.

About this edition

This is pre-release documentation for Studio 1.0.0. The PDF and online reference are generated from one reviewed language catalog. A compiler build is not a hardware, firmware or host-behavior certification.

Use automation only on computers you own or have permission to test. Read each source example and its stated effect before running it on a device.

The PDF uses embedded Geist and IBM Plex Mono fonts, with JetBrains Mono for missing Greek glyphs. These families are licensed under the SIL Open Font License 1.1. Complete font notices and provenance are bundled with the guide's maintainer sources.

This A4 document uses white pages, vector text and artwork, and black-only body text. Print at actual size or fit to your printer's printable area. The lime accent is a process-color approximation; the guide also works in grayscale.

Use search, bookmarks and the linked index to find entries in this PDF. Prefer the online guide for keyboard navigation and screen-reader access.

READ ONLINE

<https://studio.plugnpwn.io/learn/>

ORIGINAL EXAMPLE SOURCE

Source example license

MIT License

Copyright (c) 2026 PlugnPwn contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PlugnPwn name, logos and mascot artwork are excluded from this software license. Fonts and third-party dependencies retain their own licenses; see public/THIRD-PARTY-NOTICES.txt and public/fonts/OFL-Geist.txt.

EVERY ENTRY / A-Z

Find a command.

Select an entry or its page number to jump to the full reference. Reserved device variables appear under `$_`; helper source listings are included under their function names.

<code>\$_BUTTON_ENABLED</code>	49	<code>\$_RANDOM_NUMBER_KEYCODE</code>	69
<code>\$_BUTTON_PUSH_RECEIVED</code>	50	<code>\$_RANDOM_SEED</code>	64
<code>\$_BUTTON_TIMEOUT</code>	50	<code>\$_RANDOM_SPECIAL_KEYCODE</code>	69
<code>\$_BUTTON_USER_DEFINED</code>	49	<code>\$_RANDOM_UINT16</code>	64
<code>\$_CAPSLOCK_ON</code>	53	<code>\$_RANDOM_UPPER_LETTER_KEYCODE</code>	68
<code>\$_CURRENT_ATTACKMODE</code>	60	<code>\$_RECEIVED_HOST_LOCK_LED_REPLY</code>	56
<code>\$_CURRENT_PID</code>	59	<code>\$_SAVED_CAPSLOCK_ON</code>	55
<code>\$_CURRENT_VID</code>	58	<code>\$_SAVED_NUMLOCK_ON</code>	55
<code>\$_EXFIL_LEDS_ENABLED</code>	53	<code>\$_SAVED_SCROLLLOCK_ON</code>	56
<code>\$_EXFIL_MODE_ENABLED</code>	57	<code>\$_SCROLLLOCK_ON</code>	54
<code>\$_HOST_CONFIGURATION_REQUEST_COUNT</code>	60	<code>\$_STORAGE_ACTIVE</code>	62
<code>\$_INJECTING_LEDS_ENABLED</code>	52	<code>\$_STORAGE_ACTIVITY_TIMEOUT</code>	57
<code>\$_JITTER_ENABLED</code>	61	<code>\$_STORAGE_LEDS_ENABLED</code>	51
<code>\$_JITTER_MAX</code>	61	<code>\$_SYSTEM_LEDS_ENABLED</code>	51
<code>\$_LED_SHOW_STORAGE_ACTIVITY</code>	52	A first payload	5
<code>\$_NUMLOCK_ON</code>	54	Arithmetic operators	17
<code>\$_OS</code>	59	ATTACKMODE	34
<code>\$_PAYLOAD_PARSE_SPEED</code>	58	Bitwise operators	19
<code>\$_RANDOM_ASCII_CHAR</code>	67	BUTTON_DEF · END_BUTTON	37
<code>\$_RANDOM_ASCII_LETTER</code>	66	Comparison operators	18
<code>\$_RANDOM_ASCII_LOWER_LETTER</code>	65	DEBUGGER_BREAKPOINT · INJECT_BREAKPOINT_LINE_NUMBER	29
<code>\$_RANDOM_ASCII_NUMBER</code>	66	DEFAULT_DELAY · DEFAULTDELAY	43
<code>\$_RANDOM_ASCII_SPECIAL</code>	67	DEFINE	25
<code>\$_RANDOM_ASCII_UPPER_LETTER</code>	65	DELAY	40
<code>\$_RANDOM_CHAR_KEYCODE</code>	70	ENABLE_BUTTON · DISABLE_BUTTON	38
<code>\$_RANDOM_INT</code>	62	ENABLE_SYSTEM_LEDS · DISABLE_SYSTEM_LEDS	39
<code>\$_RANDOM_LETTER_KEYCODE</code>	70	EXFIL	46
<code>\$_RANDOM_LOWER_LETTER_KEYCODE</code>	68	Expression grouping	19
<code>\$_RANDOM_MAX</code>	63	EXTENSION · VERSION · END_EXTENSION	28
<code>\$_RANDOM_MIN</code>	63		

FUNCTION · END_FUNCTION	22	STRINGLN	9
HIDE_PAYLOAD · RESTORE_PAYLOAD	47	TRANSLATE_BOOL: complete source	33
HOLD · RELEASE · RESET	13	TRANSLATE_HEX: complete source	31
IF · ELSE IF · ELSE · END_IF	21	TRANSLATE_INT: complete source	29
IF_DEFINED_TRUE · IF_NOT_DEFINED_TRUE	26	TRUE · FALSE	17
INJECT · KEYCODE	13	USB descriptor parameters	34
INJECT_MOD	12	VAR · assignment	16
INJECT_VAR	15	WAIT_FOR_* lock-state commands	44
Inspect host observations	48	WAIT_FOR_BUTTON_PRESS	36
Inter-keystroke jitter	42	WAIT_FOR_STORAGE_ACTIVITY · WAIT_FOR_STORAGE_INACTIVITY	45
KEY_DOWN / UP · MOD_DOWN / UP · MOD_KEY_DOWN / UP	14	WHILE · END_WHILE	22
Keyboard layout	6		
LED_R · LED_G · LED_OFF	38		
Lock-state capture	47		
Logical operators	18		
Modifier combinations	11		
Named keys & aliases	11		
Operating-system constants	20		
OS_DETECT · TRANSLATE source libraries	28		
Random descriptor parameters	35		
RANDOM_* character commands	40		
REM · REM_BLOCK · END_REM	7		
REPEAT	27		
Reserved-variable aliases	71		
RETURN	23		
Runtime random numbers	41		
SAVE_ATTACKMODE · RESTORE_ATTACKMODE	36		
SAVE_HOST_KEYBOARD_LOCK_STATE · RESTORE_HOST_KEYBOARD_LOCK_STATE	45		
STAGE · END_STAGE	27		
STOP_PAYLOAD · RESTART_PAYLOAD	24		
STRING	8		
STRING blocks · STRINGLN blocks	9		
STRING_<LANG> · STRINGLN_<LANG>	10		
STRING_DELAY	42		