# The Button

## Overview

The button on the USB Rubber Ducky can be used to control payload execution.

By default, if no button definition (`BUTTON_DEF`) has yet been defined in the payload, pressing the button will invoke `ATTACKMODE STORAGE`. This will disable any further keystroke injection and effectively turning the USB Rubber Ducky into a mass storage flash drive, often referred to as "Arming Mode".

## WAIT_FOR_BUTTON_PRESS

Halts payload execution until a button press is detected.

When this command is reached in the payload, no further execution will occur until the button is pressed. Additionally, while waiting for the button to be pressed, the button definition (either set using `BUTTON_DEF` or the default) will be temporarily suppressed.

### #Example

```
STRING Press the button...
WAIT_FOR_BUTTON_PRESS
STRING The button was pressed!
```

**Result**

The text "The button was pressed!" will not be typed until the button is pressed.

### #Example

```
STRING Press the button 3 times...
WAIT_FOR_BUTTON_PRESS
STRING 1...
WAIT_FOR_BUTTON_PRESS
STRING 2...
```

```
WAIT_FOR_BUTTON_PRESS
STRING 3... You did it!
```

**Result**

The button must be pressed 3 times to complete the payload.

### #Example

```
LED_R
REM First Stage Payload Code...

REM Wait for operator to assess target before executing second s
WAIT_FOR_BUTTON_PRESS

LED_G
REM Second Stage Payload Code...
```

**Result**

The operator is instructed to press the button as soon as the target is ready for the next stage.

The `LED` command is used to indicate to the operator that the payload is waiting for a button press.

## BUTTON_DEF

`BUTTON_DEF` defines a special function which will execute when the button is pressed anytime throughout the payload (as long as the button control is not already in use by the `WAIT_FOR_BUTTON_PRESS` command).

By default, if no button definition (`BUTTON_DEF`) has been defined in your payload at the time the button is pressed, the button will stop all further payload execution and invoke `ATTACKMODE STORAGE` — entering the USB Rubber Ducky into arming mode.

Similar to functions (described later), which begin with `FUNCTION NAME()` and end with `END_FUNCTION`, the button definition begins with `BUTTON_DEF` and ends with `END_BUTTON`.

### #Example

```
BUTTON_DEF
  STRING The button was pressed!
  STOP_PAYLOAD
END_BUTTON

WHILE TRUE
  STRING .
  DELAY 1000
END WHILE
```

**Result**

The payload will type a period every second until the button is pressed.

Once the button is pressed, the payload will type the text "The button was pressed!"

After the button press text is typed, the payload will terminate.

## #Example

```
BUTTON_DEF
  WHILE TRUE
  LED_R
  DELAY 1000
  LED_OFF
  DELAY 1000
  END_WHILE
END_BUTTON

STRING Press the button at any point to blink the LED red
WHILE TRUE
  STRING .
  DELAY 1000
END WHILE
```

**Result**

If the button is pressed at any point in the payload it will stop typing "`.`" and the LED will start blink red until the device is unplugged.

### #Example

```
BUTTON_DEF
  REM This is the first button definition
  STRINGLN The button was pressed once!
  BUTTON_DEF
    REM This second button definition overwrites the first
    STRINGLN The button was pressed twice!
  END_BUTTON
END_BUTTON

STRINGLN Press the button twice to see how nested button definit
WHILE TRUE
  STRING .
  DELAY 1000
END WHILE
```

**Result**

If the button is pressed once at any point in the payload it will stop typing "`.`" and the first button definition will be executed.

When the first button definition is executed, a secondary button definition will be implemented.

If the button pressed a second time, the newly implement second button definition will execute.

## DISABLE_BUTTON

The `DISABLE_BUTTON` command prevents the button from calling the `BUTTON_DEF`.

### #Example

```
BUTTON_DEF
  STRING This will never execute
END_BUTTON

DISABLE_BUTTON

STRING The button is disabled
WHILE TRUE
  STRING .
  DELAY 1000
END_WHILE
```

**Result**

The `DISABLE_BUTTON` command disables the button.

The button definition which would inject "`This will never execute`", will never execute — even if the button is pressed.

### #Example

```
ATTACKMODE_OFF
LED_OFF
DISABLE_BUTTON
```

**Result**

The USB Rubber Ducky will be effectively disabled.

## ENABLE_BUTTON

The `ENABLE_BUTTON` command allows pressing the button to call the `BUTTON_DEF`.

**Example**

```
BUTTON_DEF
  STRINGLN The button was pressed!
  STRINGLN Continuing the payload...
```

```
END_BUTTON

WHILE TRUE
  DISABLE_BUTTON
  STRINGLN The button is disabled for the next 5 seconds...
  STRINGLN Pressing the button will do nothing...
  DELAY 5000

  ENABLE_BUTTON
  STRINGLN The button is enabled for the next 5 seconds...
  STRINGLN Pressing the button will execute the button definiti
  DELAY 5000
END_WHILE
```

**Result**

The payload will alternate between the button being enabled and disabled.

If the button is pressed within the 5 second disabled window, nothing will happen.

If the button is pressed within the 5 second enabled window, the button definition will be executed and "`The button was pressed!`" will be typed.

The payload will loop forever.

## #Example

```
BUTTON_DEF
  STRINGLN The button was pressed!
  DISABLE_BUTTON
  STRINGLN Pressing the button again will do nothing.
END_BUTTON

STRING Press the button at any time to execute the button defin
WHILE TRUE
  STRING .
END_WHILE
```

**Result**

The payload will continuously type "`.`".

Pressing the button will execute the `BUTTON_DEF`.

The `BUTTON_DEF` will type "`The button was pressed!`", then disable itself with the `DISABLE_BUTTON` command. This will be announced by typing the message "`Pressing the button again will do nothing.`"

The payload will continue to type "`.`" as before.

Pressing the button again will do nothing.

## Internal Variables

The following internal variables relate to the button operation and may be used in your payload for advanced functions.

### $_BUTTON_ENABLED

Returns `TRUE` if the button is enabled or `FALSE` if the button is disabled.

```
IF ($_BUTTON_ENABLED == TRUE) THEN
    REM The button is enabled
ELSE IF ($_BUTTON_ENABLED == FALSE) THEN
    REM The button is disabled
END_IF
```

May be set `TRUE` to enabled the button or `FALSE` to disable the button.

```
$_BUTTON_ENABLED = TRUE
```

## $_BUTTON_USER_DEFINED

Returns `TRUE` if a `BUTTON_DEF` has been already defined in the payload or `FALSE` if it hasn't and is still the default.

```
IF ($_BUTTON_USER_DEFINED == TRUE) THEN
    REM Pressing the button will run the user defined BUTTON_DEI
END_IF
```

May be set `TRUE` or `FALSE`, however caution must be taken as setting `TRUE` when a `BUTTON_DEF` does not exist will cause undefined behavior.

```
BUTTON_DEF
    $_BUTTON_USER_DEFINED = FALSE
    REM Pressing the button will disable the button definition
END_BUTTON
```

## $_BUTTON_PUSH_RECEIVED

Returns `TRUE` if the button has ever been pressed.

This variable may be retrieved or set.

```
REM Example $_BUTTON_PUSH_RECEIVED

STRING PUSH BUTTON N times within 5s
$CD = 15

WHILE ($CD > 0)
  IF ($_BUTTON_PUSH_RECEIVED == TRUE) THEN
    STRINGLN Passed
    $_BUTTON_PUSH_RECEIVED = FALSE
  END_IF

  $CD = ($CD - 1)
  STRING .
  DELAY 200
END_WHILE
```

```
$_BUTTON_ENABLED = TRUE
$_BUTTON_PUSH_RECEIVED = FALSE
```

## $_BUTTON_TIMEOUT

The button debounce, or cooldown time before counting the next button press, in milliseconds.

The default value is 1000.