LOOPS

Overview

Loops are flow control statements that can be used to repeat instructions until a specific condition is reached.

WHILE

A block of code can be executed repeatedly a specified number of times (called iterations) using a `WHILE` statement. The code within the `WHILE` statement will continue to execute for as long as the condition of the `WHILE` statement is `TRUE`.

A `WHILE` statement is similar to an `IF` statement, however it behaves differently when at the statements end. Whereas at the end of an `IF` statement the payload will continue, when the end of a `WHILE` statement is reached the payload execution will jump back to the beginning of the WHILE statement and reevaluate the condition. One way to interpret a `WHILE` statement is to read it as "IF this condition is true, THEN do that until it isn't true anymore" — hence it being called a while loop.

Syntax

The `WHILE` statement consists of four parts

1. The `WHILE` keyword.
2. The condition, or expression that evaluates to `TRUE` or `FALSE`.
3. One or more newlines containing the block of code to execute.
4. The `END_WHILE` keyword.

Example

```
REM Example while loop - blink LED 42 times

VAR $FOO = 42
WHILE ( $FOO > 0 )
    LED_G
    DELAY 500
    LED_OFF
    DELAY 500
    $FOO = ( $FOO - 1 )
END_WHILE

LED_R
```

Result

- The variable `$FOO` is set to 42.
- The `WHILE` loop begins, evaluating the condition "is $FOO greater than 0".
- Every time the condition is `TRUE`, the block of code between `WHILE` and `END_WHILE` will run.The LED will blink green: half a second on, half a second off.The variable `$FOO` will decrement by one.
- Once `$FOO` reaches zero, the `WHILE` condition will no longer evaluate to `TRUE`. The payload will continue execution after the `END_WHILE` statement, where the LED will light red.
- If the button is pressed at any time during the payload execution, the `WHILE` loop will end and the USB Rubber Ducky will enter `ATTACKMODE STORAGE` since that is the default behavior when no `BUTTON_DEF` has been initiated.

Example

```
REM Example while loop - press the button 5 times

VAR $FOO = 5

WHILE ( $FOO > 0 )
    STRINGLN Press the button...
    WAIT_FOR_BUTTON_PRESS
    $FOO = ( $FOO - 1 )
END_WHILE
```

```
STRINGLN You pressed the button 5 times!
```

Result

- The variable `$FOO` is set to 5.
- The code block within the `WHILE` loop will be repeated until the expression evaluates to `FALSE`.
- For each run of the code block, the message "`Press the button...`" is typed. The payload then waits until it detects the button is pressed, at which point the variable `$FOO` is decremented.

Infinite Loop

The syntax of `WHILE` states that in nearly all cases the expression should be surrounded by parenthesis `( )`. The exception is when initiating an infinite loop. The condition of the expression `TRUE` will always evaluate to `TRUE`. While it is not necessary to omit the parenthesis, it is technically more efficient. This is because it directly references `TRUE`, reducing the number of instructions and removing the step of first reducing the order of precedence.

This is loop that will execute endlessly, until intervention occurs. This may either come in the form of a button press, or simply by unplugging the USB Rubber Ducky.

Example

```
REM Example Infinite Loop

BUTTON_DEF
    WHILE TRUE
        LED_R
        DELAY 500
        LED_OFF
        DELAY 500
    END_WHILE
END_BUTTON
```

```
WHILE TRUE
    LED_G
    DELAY 500
    LED_OFF
    DELAY 500
END_WHILE
```

Result

- Because a button definition has been initiated with `BUTTON_DEF`, the default behavior will no longer apply when the button is pressed.
- The LED will blink green: half a second on, half a second off.
- Pressing the button will stop the currently infinite loop of blinking the LED green and execute the button definition, thus blinking the LED red.