# Guide to PROGMEM on ESP8266 and Arduino IDE

## Intro

PROGMEM is a Arduino AVR feature that has been ported to ESP8266 to ensure compatability with existing Arduino libraries, as well as, saving RAM. On the esp8266 declaring a string such as const char \* xyz = "this is a string" will place this string in RAM, not flash. It is possible to place a String into flash, and then load it into RAM when it is needed. On an 8bit AVR this process is very simple. On the 32bit ESP8266 there are conditions that must be met to read back from flash.

On the ESP8266 PROGMEM is a macro:

*#define PROGMEM ICACHE\_RODATA\_ATTR*

ICACHE\_RODATA\_ATTR is defined by:

*#define ICACHE\_RODATA\_ATTR \_\_attribute\_\_((section(".irom.text")))*

Which places the variable in the .irom.text section ie flash. Placing strings in flash requires using any of the methods above.

### Declare a global string to be stored in flash.

static const char xyz[] PROGMEM **=** "This is a string stored in flash";

## Declare a flash string within code block.

For this you can use the PSTR macro. Which are all defined in[pgmspace.h](https://github.com/esp8266/Arduino/blob/master/cores/esp8266/pgmspace.h)

*#define PGM\_P const char \**

*#define PGM\_VOID\_P const void \**

*#define PSTR(s) (\_\_extension\_\_({static const char \_\_c[] PROGMEM = (s); &\_\_c[0];}))*

In practice:

void myfunction(void) {

PGM\_P xyz **=** PSTR("Store this string in flash");

const char **\*** abc **=** PSTR("Also Store this string in flash");

}

The two examples above will store these strings in flash. To retrieve and manipulate flash strings they must be read from flash in 4byte words. In the Arduino IDE for esp8266 there are several functions that can help retrieve strings from flash that have been stored using PROGMEM. Both of the examples above return const char \*. However use of these pointers, without correct 32bit alignment you will cause a segmentation fault and the ESP8266 will crash. You must read from the flash 32 bit aligned.

## Functions to read back from PROGMEM

Which are all defined in [pgmspace.h](https://github.com/esp8266/Arduino/blob/master/cores/esp8266/pgmspace.h)

int memcmp\_P(const void**\*** buf1, PGM\_VOID\_P buf2P, size\_t size);

void**\*** memccpy\_P(void**\*** dest, PGM\_VOID\_P src, int c, size\_t count);

void**\*** memmem\_P(const void**\*** buf, size\_t bufSize, PGM\_VOID\_P findP, size\_t findPSize);

void**\*** memcpy\_P(void**\*** dest, PGM\_VOID\_P src, size\_t count);

char**\*** strncpy\_P(char**\*** dest, PGM\_P src, size\_t size);

char**\*** strcpy\_P(dest, src)

char**\*** strncat\_P(char**\*** dest, PGM\_P src, size\_t size);

char**\*** strcat\_P(dest, src)

int strncmp\_P(const char**\*** str1, PGM\_P str2P, size\_t size);

int strcmp\_P(str1, str2P)

int strncasecmp\_P(const char**\*** str1, PGM\_P str2P, size\_t size);

int strcasecmp\_P(str1, str2P)

size\_t strnlen\_P(PGM\_P s, size\_t size);

size\_t strlen\_P(strP)

char**\*** strstr\_P(const char**\*** haystack, PGM\_P needle);

int printf\_P(PGM\_P formatP, **...**);

int sprintf\_P(char **\***str, PGM\_P formatP, **...**);

int snprintf\_P(char **\***str, size\_t strSize, PGM\_P formatP, **...**);

int vsnprintf\_P(char **\***str, size\_t strSize, PGM\_P formatP, va\_list ap);

There are a lot of functions there but in reality they are \_Pversions of standard c functions that are adapted to read from the esp8266 32bit aligned flash. All of them take a PGM\_P which is essentially a const char \*. Under the hood these functions all use, a process to ensure that 4 bytes are read, and the request byte is returned.

This works well when you have designed a function as above that is specialised for dealing with PROGMEM pointers but there is no type checking except against const char \*. This means that it is totally legitimate, as far as the compiler is concerned, for you to pass it any const char \* string, which is obviously not true and will lead to undefined behaviour. This makes it impossible to create any overloaded functions that can use flash strings when they are defined as PGM\_P. If you try you will get an ambiguous overload error as PGM\_P == const char \*.

Enter the \_\_FlashStringHelper… This is a wrapper class that allows flash strings to be used as a class, this means that type checking and function overloading can be used with flash strings. Most people will be familiar with the F() macro and possibly the FPSTR() macro. These are defined in [WString.h](https://github.com/esp8266/Arduino/blob/master/cores/esp8266/WString.h" \l "L37):

*#define FPSTR(pstr\_pointer) (reinterpret\_cast<const \_\_FlashStringHelper \*>(pstr\_pointer))*

*#define F(string\_literal) (FPSTR(PSTR(string\_literal)))*

So FSPTR() takes a PROGMEM pointer to a string and casts it to this \_\_FlashStringHelper class. Thus if you have defined a string as above xyz you can use FPSTR() to convert it to\_\_FlashStringHelper for passing into functions that take it.

static const char xyz[] PROGMEM **=** "This is a string stored in flash";

Serial**.**println(FPSTR(xyz));

The F() combines both of these methods to create an easy and quick way to store an inline string in flash, and return the type \_\_FlashStringHelper. For example:

Serial**.**println(F("This is a string stored in flash"));

Although these two functions provide a similar function, they serve different roles. FPSTR() allows you to define a global flash string and then use it in any function that takes \_\_FlashStringHelper. F() allows you to define these flash strings in place, but you can’t use them anywhere else. The consequence of this is sharing common strings is possible using FPSTR() but not F(). \_\_FlashStringHelper is what the String class uses to overload its constructor:

String(const char **\***cstr **=** ""); **//** constructor **from** const char **\***

String(const String **&**str); **//** copy constructor

String(const \_\_FlashStringHelper **\***str); **//** constructor **for** flash strings

This allows you to write:

String mystring(F("This string is stored in flash"));

How do I write a function to use \_\_FlashStringHelper? Simples: cast the pointer back to a PGM\_P and use the \_P functions shown above. This an example implementation for String for the concat function.

unsigned char String::concat(const \_\_FlashStringHelper \* str) {

if (!str) return 0; // return if the pointer is void

int length = strlen\_P((PGM\_P)str); // cast it to PGM\_P, which is basically const char \*, and measure it using the \_P version of strlen.

if (length == 0) return 1;

unsigned int newlen = len + length;

if (!reserve(newlen)) return 0; // create a buffer of the correct length

strcpy\_P(buffer + len, (PGM\_P)str); //copy the string in using strcpy\_P

len = newlen;

return 1;

}

## How do I declare a global flash string and use it?

static const char xyz[] PROGMEM **=** "This is a string stored in flash. Len = %u";

void setup() {

Serial**.**begin(115200); Serial**.**println();

Serial**.**println( FPSTR(xyz) ); **//** just prints the string, must convert it to FlashStringHelper first using FPSTR()**.**

Serial**.**printf\_P( xyz, strlen\_P(xyz)); **//** use printf **with** PROGMEM string

}

## How do I use inline flash strings?

void setup() {

Serial**.**begin(115200); Serial**.**println();

Serial**.**println( F("This is an inline string")); **//**

Serial**.**printf\_P( PSTR("This is an inline string using printf %s"), "hello");

}

## How do I declare and use data in PROGMEM?

const size\_t len\_xyz **=** 30;

const uint8\_t xyz[] PROGMEM **=** {

0x53, 0x61, 0x79, 0x20, 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20,

0x74, 0x6f, 0x20, 0x4d, 0x79, 0x20, 0x4c, 0x69, 0x74, 0x74,

0x6c, 0x65, 0x20, 0x46, 0x72, 0x69, 0x65, 0x6e, 0x64, 0x00};

void setup() {

Serial**.**begin(115200); Serial**.**println();

uint8\_t **\*** buf **=** new uint8\_t[len\_xyz];

**if** (buf) {

memcpy\_P(buf, xyz, len\_xyz);

Serial**.**write(buf, len\_xyz); **//** output the buffer**.**

}

}

## How do I declare some data in PROGMEM, and retrieve one byte from it.

Declare the data as done previously, then use pgm\_read\_byte to get the value back.

const size\_t len\_xyz **=** 30;

const uint8\_t xyz[] PROGMEM **=** {

0x53, 0x61, 0x79, 0x20, 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x20,

0x74, 0x6f, 0x20, 0x4d, 0x79, 0x20, 0x4c, 0x69, 0x74, 0x74,

0x6c, 0x65, 0x20, 0x46, 0x72, 0x69, 0x65, 0x6e, 0x64, 0x00

};

void setup() {

Serial**.**begin(115200); Serial**.**println();

**for** (int i **=** 0; i **<** len\_xyz; i**++**) {

uint8\_t byteval **=** pgm\_read\_byte(xyz **+** i);

Serial**.**write(byteval); **//** output the buffer**.**

}

}

## In summary

It is easy to store strings in flash using PROGMEM and PSTR but you have to create functions that specifically use the pointers they generate as they are basically const char \*. On the other hand FPSTR and F() give you a class that you can do implicit conversions from, very useful when overloading functions, and doing implicit type conversions. It is worth adding that if you wish to store an int, float or pointer these can be stored and read back directly as they are 4 bytes in size and therefor will be always aligned!

Hope this helps.