

- [Home](#)
- [Libraries](#)
  - [TinyGPS](#)
  - [NewSoftSerial](#)
  - [Streaming](#)
  - [PString](#)
  - [Flash](#)
  - [PWMServo](#)
- [About](#)
- [Projects](#)
  - [The Reverse Geocache™ Puzzle](#)
    - [Building a Puzzle Box](#)
    - [More Puzzle Box Tales](#)

Email:  [Subscribe!](#) [Admin RSS](#) Subscribe: [RSS feed](#)

[Arduiniana](#)

Arduino wisdom and gems by Mikal Hart

## Flash

### A Library to Ease Accessing Flash-based (PROGMEM) Data

Storing static program data in flash/PROGMEM is a tricky part of Arduino programming. To save precious RAM, a novice user already at odds with unfamiliar C++ syntax must digest such daunting concepts as `prog_char`, `PSTR()`, `PROGMEM`, `pgm_read_word()`, etc. Even seasoned users get tripped up by the indirection and typecasting that are required to retrieve valid PROGMEM data. Add to that a couple of apparent bugs in the implementation, and it's clear that PROGMEM is a complicated mess.

I have written a new library, **Flash**, which abstracts away most of this complexity. It provides new **String**, **Array**, **Table**, and **String Array** types that make ROM-based data collections as easy to use as “normal” types. Each overrides the C++ `[]` operator, so to extract individual elements one uses familiar array access syntax:

```
// float array example
float t = temperatures[950];

// (2D) font table example
digitalWrite(pin[i], font_table['a'][i]);

// string example
for (int i=0; i<str.length(); ++i)
    if (str[i] == ';') break;
```

### Creating named Flash objects

To create a Flash object, you use a library-provided macro. Each of the four basic types provides its own creation macro:

**Strings:** `FLASH_STRING(name, value)`

**Arrays:** `FLASH_ARRAY(type, name, list of values...)`

**Tables:** `FLASH_TABLE(type, name, columns, values...)`

**String Arrays:** `FLASH_STRING_ARRAY(name, values...)`

Examples:

```
FLASH_STRING(big_string, "Moreover, as you might appreciate, "
  "their implications were such as to provoke a certain "
  "degree of sorrow within me. Indeed - why should I not "
  "admit it? - at that moment, my heart was breaking.");

FLASH_ARRAY(float, temperatures, 23.1, 23.1, 23.2, 23.2, 23.4,
  23.7, 25.0, 26.0, 26.8, 28.8, 30.2, 31.9, 33.1, 33.1, 33.2,
  33.2, 33.4, 33.7, 35.0, 36.0, 36.8, 38.8, 40.2, 41.9);

FLASH_TABLE(boolean, font_table, 7, {0,0,0,0,0,0,0},
  {0,0,0,1,0,1,1}, {1,0,1,1,1,0,1}, {1,1,0,0,0,0,0},
  {0,1,0,1,0,1,0}, {1,0,1,1,1,0,1}, {1,0,0,1,0,0,1},
  {0,0,1,1,0,1,1}, {1,0,1,1,1,1,1});

FLASH_STRING_ARRAY(digits, PSTR("Zero"), PSTR("One"),
  PSTR("Two"), PSTR("Three"), PSTR("Four"), PSTR("Five"),
  PSTR("Six"), PSTR("Seven"), PSTR("Eight"), PSTR("Nine"));
```

[Note that you can make Arrays and Tables out of any native type.]

[Note also that the **String Array** is a slightly special case. For technical reasons, as you can see in the example, each member of the array initializer must be wrapped with the PSTR() operator. For this reason, String Arrays cannot be declared at global scope, but must be non-statically defined within in the body of a function. Note also that while the individual strings in a String Array are contained in PROGMEM space, the array itself (again, for technical reasons) is RAM-based, consuming 2 bytes per array element. This limitation does not apply to the other Flash types.]

## Using named Flash objects

**Array-like access.** Each Flash type provides an [] operator to get access to the underlying data, for example:

```
// extract an element from an Array
float t = temperatures[5];

// examine characters in a String
for (int i=0; i<big_string.length(); ++i)
  if (big_string[i] == 'O') ++big_O_counter;

// Use a font table to configure a 7-segment display
for (int i=0; i<font_table[1].count(); ++i)
  digitalWrite(pins[i], font_table[1][i]);

// Examine the sizes of strings in a String Arra
for (int i=0; i<digits.count(); ++i)
  Serial.println(digits[i].length());
```

Since **Tables** are really just two-dimensional arrays, the **Table** object's implementation of [] returns a one-dimensional **Array** object.

**Size.** Each Flash type provides a mechanism for determining its size:

String and String Array: size\_t **length()**;

Array: size\_t **count()**;

Table: size\_t **rows()**; size\_t **cols()**;

**Access.** If you need it, each **Flash** object provides an **access()** method that returns the underlying PROGMEM pointer

**Print compatibility.** Each **Flash** object has a **print()** method that can be used to serialize the object to any stream derived from core class **Print** (HardwareSerial, NewSoftSerial, LiquidCrystal, Ethernet, PString, etc.) See "Printing Flash objects" below.

**String copy.** Flash String objects may be copied in whole or part to RAM using the String object's copy method:

```
// copies the first 10 characters of the big_string
char temp[11];
big_string.copy(temp, 10);
```

## Printing Flash objects

Each **Flash** object can “print” itself, using either its print method directly:

```
temperatures.print(Serial);
big_string.print(lcd);
font_table.print(newsoftserial);
```

or, equivalently, by using the << streaming operator:

```
Serial << temperatures;
lcd << big_string;
newsoftserial << font_table;
```

**Flash** objects can also be “printed” using the familiar **Serial.print[ln]** syntax, but this requires a small patch to the core Print.h file. See “Patching Print.h” below

## Using unnamed (inline) Flash Strings

One of the most useful features of the **Flash** library is the capability to print unnamed strings embedded directly within an expression. In other words, if you want to use a ROM string in a print statement, it is not necessary to declare a string variable and print it like this:

```
FLASH_STRING(big_string12,
  "'Twas brillig and the slithy toves did gyre and gimble");
lcd << big_string12;
```

Instead you can simply use the new built-in F() macro (F for “Flash”) to do the same thing inline:

```
lcd << F("'Twas brillig and the slithy toves did gyre and gimble");
```

And again, you can also use

```
lcd.print(F("'Twas brillig and the slithy toves did gyre and gimble"));
```

to print an inline string, but this requires the patch to Print.h.

## Patching Print.h (optional)

By making a very small change to the core Print.h file, you can add the capability to print either named or unnamed **Flash** objects using the familiar **print[ln]** syntax. What the patch does is tell class Print about **Flash** objects and how to print them. Note that while this patch is useful, it is entirely optional. Flash already provides two mechanisms for printing ROM strings and data structures:

```
obj.print(stream);
stream << obj;
```

This patch adds a third technique, the familiar

```
stream.print(obj);
```

To patch, add the following interface definition to the top of Print.h, right after the #include statements. This defines what a Flash object is, from the perspective of the Print class.

```
#define ARDUINO_CORE_PRINTABLE_SUPPORT
class Print;
class _Printable
{
public:
    virtual void print(Print &stream) const = 0;
};
```

Then add a couple of inline print methods — alongside all the others — to the body of class Print:

```
void println(const _Printable &obj)
{ obj.print(*this); println(); }
void print(const _Printable &obj)
{ obj.print(*this); }
```

No need to recompile anything; you are now ready to roll:

```
Serial.print(myarray);
Serial.print(mytable);
Serial.print(F("A long, long, unnamed inline ROM string"));
```

## Library Version

You can retrieve the version of the **Flash** library by inspecting `FLASH_LIBRARY_VERSION`.

```
int ver = FLASH_LIBRARY_VERSION;
```

## Download

The latest version of **Flash** is available here: [Flash3.zip](#)

## Change Log

1. initial version
2. moved some code to a new .cpp file to avoid inlining every member function
3. fixed a bug in the copy() method

## Acknowledgements

Thanks to forum user Bill W. (“westfw”) for testing **Flash** and observing that Strings take up a surprising amount of extra PROGMEM space (20 bytes per string). I used his data to modify the library so that some of the commonly used functions, notably the `_FLASH_STRING` constructor are not inlined. This change reduces the footprint of the example programs noticeably. Thanks, Bill!

I appreciate any suggestions or input.

Mikal Hart

Page last updated on January 6, 2010 at 9:41 pm

69 Responses → “Flash”

[« Older Comments](#)

1.

brad

5 months ago

Why does this not work? and what would be a good way to implement something like returning a

string stored in ROM based on the case?

```
String identifydata(int x){ //accepts a binary number and returns a corresponding number that
matches into state["blah"] to match to a human string
//Serial.print(x); // see what binary number were trying to match
FLASH_STRING(yo,"YOO")
switch(x){
case 0 : // see if binary number matches zero
return (yo);
break;
case 1010101010:
return(1);
break;
case 1110000111:
return(2);
}
```

2.

Mikal

5 months ago

Hi Brad,

I think it would be possible to create several `_FLASH_STRING` objects and then switch on an index to return one of them. Your example fails because in the one case (0) you return a `FLASH_STRING`, but in others you return integers.

Mikal

3.

Sam

4 months ago

Is there a way to write to an individual element of a `flash_table`.

First I initiated and a table:

```
FLASH_TABLE(boolean,relay,10,
{0,0,0,0,0,0,0,0,0,0},
{0,1,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0});
```

I can read a specific element with the phrase:

```
Serial.println(relay[i][0],BIN); //[row][col]
```

I would like to write to an element in the same way although the following doesn't work:

```
boolean relay[i][1]=1;
```

4.

Mikal

4 months ago

Sorry, Sam. From the Arduino sketch's point of view, all flash-based data is strictly read-only.

Mikal

5.

Meredith Turner

3 months ago

Jumping into building a reverse geocache box and would like to use the libraries that you have generously made available.

What compiler would you recommend?

Thanks.

Meredith

6.

Peter

3 months ago

Is there any way to store variables into flash memory?

i.e:

```
int i = 2;  
int j = 9;  
int k = 3;  
FLASH_ARRAY(int, font_table2, i,j,k);
```

Thanks!

7.

[Mikal](#)

3 months ago

@Peter,

Generally the answer is no. You can only store values in the flash that are known at compile-time. However, if you adjust your code like this to make your "variables" compile-time constants, it will work:

```
static const int i = 2;  
static const int j = 9;  
static const int k = 3;  
FLASH_ARRAY(int, font_table2, i,j,k);
```

8.

**Shane**

2 months ago

Hi. Im new to coding and Arduino.

I have a project that uses a thermistor to return a temp value and print it to an LCD.

I would like to use a lookup table, that would look up a temp against the ADC value from the thermistor..

i just cant find out how to create such a table in the code, and be able to call on it to return me a temp that i can then print to the LCD.

i can easily get the ADC vs temp table such as

ADC Temp  
100 23C  
200 24C  
300 25C  
etc etc.

But i dont know how to do that in code.

Can you tell me how to do it? or point me in the right direction?

Ive searched the arduino site, and google and cant find anything that gives me a breakdown on how to create the table that i want and how to compare an ADC value and get a result.

9.

**Jan**

2 months ago

**Hi!**

Thank you for a nice lib. I'm just quite new to C and to arduino and I would need to explain one thing. Can I create a function to accept what's generated by your FLASH\_STRING\_ARRAY, FLASH\_TABLE or FLASH\_STRING macros as a parameter? Could you give me some hint how to do that? Thanks a lot!

10.

**Mikal**

2 months ago

Jan, the FLASH\_STRING macro creates and object of type \_FLASH\_STRING.

FLASH\_ARRAY creates an object of type \_FLASH\_ARRAY.

FLASH\_TABLE creates an object of type \_FLASH\_TABLE.

Mikal

11.

[Patrick](#)

1 month ago

How would I store a 2 dimension array with Flash.h??

```
char * menu[][2] = {
  {"Item 1","apples, coke" },
  {"Item 2","OJ, Cranberry" },
  {"Item 3","Peach, Bamboo" },
  {"Item 4","cheese, carrots" },
  {"Item 5","big fluffy chicken cat" }
};
```

Do I have to build two different String Arrays? Can you store char arrays inside of a Table Array??

12.

[Mikal](#)

1 month ago

Patrick, I'm afraid there's no convenient way to make a 2D array of strings.

13.

Synthy

1 month ago

Hi Mikal,

I have made a lookup string, but when i try to use it as in your example, the string is too long. There are 9 characters printed instead of 8. Here is the code:

```
FLASH_STRING(xgvoice, "GrandPnoGrndPnoKMelloGrPPianoStr");
```

```
myGLCD.setBackColor(0,0,0);
char temp_str[8] = {0};
xgvoice.copy(temp_str, 8, vc*8);
myGLCD.print(temp_str,x,y);
```

Do you know what i did wrong?

Thanks, Pim.

14.

[Mikal](#)



1 month ago

Pim,

This is a documentation problem that I need to fix. When you do `xgvoice.copy(temp_str, 8, vc*8)`, eight characters are moved to `temp_str`, but because the underlying code call `strncpy_P`, the string is not 0-terminated. (Also `temp_str` should be 9 characters long to hold the 8 characters plus the 0 terminator.) So change

```
char temp_str[8] = {0};
```

to

```
char temp_str[9] = {0};
```

I think this will solve your problem.

15.

Travis

1 month ago

Hi Mikal,

I am trying to use your Flash library to write two data tables, which are then used for interpolation (using Kerinin's Spline library); however, when I try to compile, the spline function errors out saying invalid conversion from `_FLASH_ARRAY` to `float`. Any ideas on how to fix this with your library. (I am trying to pass the two arrays to the function)

Here is the line of code:

```
Spline linearSpline(alt_a.access(),rho_a.access(),51,1);
```

Thanks!

16.

[Mikal](#)

4 weeks ago

Travis, I won't be able to guess at the problem without a definition of the Spline class. Are `alt_a` and `rho_a` `FLASH_ARRAY`s? I'm guessing that whoever wrote Spline expects memory-based arrays, so Flash Arrays will fail. ?

M

[« Older Comments](#)

3 Trackbacks For This Post

1. [New Flash library « Arduiniana](#) →  
[March 24th, 2009 → 9:59 am](#)

[...] Interested to learn more? Read all about it here! [...]

2. [vkick Blog Site » Blog Archive » How to do Timer and Save Settings?](#) →  
[June 18th, 2009 → 4:01 pm](#)

[...] <http://arduiniana.org/libraries/Flash/> Categories: Arduino Posted By: Mon Last Edit: 18 Jun 2009 @ 04 01 PM Email • Permalink Previous Responses to this post » (None) Post a Comment Click here to cancel reply. [...]

3. [Arduino net connected clock tells weather not time](#) →  
[July 1st, 2010 → 2:38 pm](#)

[...] third party libraries were used; PString to handle parsing the incoming website, Flash for memory management, and Streaming for easier coding. [...]

### Leave a Reply

<input type="text"/>	Name *
<input type="text"/>	Mail *
<input type="text"/>	Website

Add Comment

### Pages

- [About](#)
- [Libraries](#)
  - [Flash](#)
  - [NewSoftSerial](#)
  - [PString](#)
  - [PWMServo](#)
  - [Streaming](#)
  - [TinyGPS](#)
- [Projects](#)
  - [The Reverse Geocache™ Puzzle](#)
    - [Building a Puzzle Box](#)
    - [More Puzzle Box Tales](#)

### Recent Posts

- [Engagements: A Tale of Two Cities](#)
- [Welcome, MAKE readers!](#)
- [NewSoftSerial 11 \(beta\)](#)
- [Puzzle Box 4: A Long Awaited Opening](#)
- [The Reverse Geocache™ Puzzle Box hits YouTube](#)

## Archives

- [January 2011](#)
- [October 2010](#)
- [August 2010](#)
- [July 2010](#)
- [February 2010](#)
- [January 2010](#)
- [November 2009](#)
- [October 2009](#)
- [May 2009](#)
- [April 2009](#)
- [March 2009](#)
- [February 2009](#)
- [January 2009](#)

© 2011 Mikal Hart.