# SPIMemory Documentation

***Release 0***

**Prajwal Bhattaram**

**Jun 03, 2019**

# Contents

**Important:**

**Status of documentation**

In progress (16.05.2018 @ 2045 hrs AEST)

**Status of library code**

# CHAPTER 1

# About SPIMemory

The SPIMemory library project aims to be a single unified Arduino library that allows for communication with a variety of data storage ICs that communicate via the SPI protocol.

**Current status:**

In its current form (as of v3.2.0), this library supports read/write/erase functions on a number of SPI Flash memory ICs through its `SPIFlash` class.

**In the works:**

Work on a new class `SPIFRAM`, is currently in progress and when released it will support read/write/erase functions on a number of SPI FRAM memory ICs.

**Ideas for future development:**

Support for SPI EEPROMs

CHAPTER 2

# Installation

## 2.1 Option 1

- Open the Arduino IDE.
- Go to Sketch > Include Library > Manage libraries.
- Search for SPIMemory.
- Install the latest version.

## 2.2 Option 2

- Click here
- Unzip the archive that downloads and rename resulting folder to 'SPIMemory'
- Move the folder to your libraries folder (~/sketches/libraries)

CHAPTER 3

# Compatibility

## 3.1 Arduino IDEs supported (actually tested with)

- IDE v1.5.x
- IDE v1.6.0-v1.6.5
- IDE v1.6.9-v1.6.12
- IDE v1.8.1-v1.8.5

### 3.1.1 Microcontrollers (Boards tested)

#### Completely supported

- ATmega328P (Arduino Uno, Arduino Micro, Arduino Fio, Arduino Nano)
- ATmega32u4 (Arduino Leonardo, Arduino Fio v3)
- ATmega2560 (Arduino Mega)
- ATSAMD21G18 ARM Cortex M0+ (Adafruit Feather M0, Adafruit Feather M0 Express)
- AT91SAM3X8E ARM Cortex M3 (Arduino Due)
- ATSAMD51J19 ARM Cortex M4 (Adafruit Metro M4)
- STM32F091RCT6 (Nucleo-F091RC)
- ESP8266 Boards (Adafruit ESP8266 Feather)
- Simblee Boards (Sparkfun Simblee)

### In BETA

- ESP32 Boards (Adafruit ESP32 Feather) The library is known to work with the ESP32 core as of the current commit 25dff4f on 05.04.2018.*[0]†[0]

## 3.1.2 Flash memory chips

### Completely supported (Actually tested with)

- Winbond
  - W25Q16BV
  - W25Q64FV
  - W25Q80BV
  - W25Q256FV
- Microchip
  - SST25VF064C
  - SST26VF064B
- Cypress/Spansion
  - S25FL032P
  - S25FL116K
  - S25FL127S
- ON Semiconductor
  - LE25U40CMC
- AMIC
  - A25L512A0
- Micron
  - M25P40
- Adesto
  - AT25SF041

### Should work with (Similar enough to the ones actually tested with)

- Winbond (All SPI Flash chips)
- Microchip (SST25 & SST26 series)
- Cypress/Spansion (S25FL series)
- Any flash memory that is compatible with the SFDP standard as defined in JESD216B

[0] ESP32 support will remain in beta till the ESP32 core can be installed via the Arduino boards manager.

[0] ESP32 boards usually have an SPI Flash already attached to their SS pin, so the user has to declare the ChipSelect pin being used when the constructor is declared - for example: `SPIFlash flash(33);`

# CHAPTER 4

## About SPIFlash

The SPIFlash class of the SPIMemory library is for use with flash memory chips that communicate using the SPI protocol. In its current form it supports identifying the flash chip and its various features; automatic address allocation and management; writing and reading a number of different types of data, ranging from 8-bit to 32-bit (signed and unsigned) values, floats, Strings, arrays of bytes/chars and structs to and from various locations; sector, block and chip erase; and powering down for low power operation.

More information about the API and using it can be found *here*, or simply se the table of contents on the left.

CHAPTER 5

# API and Usage

## 5.1 Library structure

### 5.1.1 Constructor:

**class SPIFlash**

#### Library instantiation functions:

bool **begin** (uint32_t *flashChipSize* = 0)

void **setClock** (uint32_t *clockSpeed*)

#### Chip ID functions:

uint16_t **getManID** (void)

uint32_t **getJEDECID** (void)

uint64_t **getUniqueID** (void)

uint32_t **getCapacity** (void)

uint32_t **getMaxPage** (void)

#### Read functions:

`Data type-independent`

bool **readAnything** (uint32_t *_addr*, T *&data*, bool *fastRead* = false)

`Data type-dependent`

uint8_t **readByte** (uint32_t *_addr*, bool *fastRead* = false) int8_t **readChar** (uint32_t *_addr*, bool *fastRead* = false)

int16_t **readShort** (uint32_t *_addr*, bool *fastRead* = false)

uint16_t **readWord** (uint32_t *_addr*, bool *fastRead* = false)

int32_t **readLong** (uint32_t *_addr*, bool *fastRead* = false)

uint32_t **readULong** (uint32_t *_addr*, bool *fastRead* = false)

float **readFloat** (uint32_t *_addr*, bool *fastRead* = false)

bool **readStr** (uint32_t *_addr*, String &*data*, bool *fastRead* = false)

bool **readByteArray** (uint32_t *_addr*, uint8_t **data_buffer*, size_t *bufferSize*, bool *fastRead* = false)

bool **readCharArray** (uint32_t *_addr*, char **data_buffer*, size_t *buffer_size*, bool *fastRead* = false)

**class SPIFlash**

*SPIFlash*::**SPIFlash** (uint8_t *cs* = CS, SPIClass **spiinterface* = &SPI)

## 5.2 Constructor `Mandatory`

**Note:** A constructor is a special kind of class member function that is executed when an object of that class is instantiated. Constructors are typically used to initialize member variables/functions of the class to appropriate default values, or to allow the user to easily initialize those member variables/functions to whatever values are desired.*[0]

### 5.2.1 Parameters `Optional`

uint8_t **cs**

Refer to *Defining a custom Chip Select pin*

SPIClass ***spiinterface**

Refer to *Using a non-default SPI interface*

### 5.2.2 What it does

#### Returns `Nothing`

- The constructor must be called before `void setup()`. The constructor can be any word of your choice. For example, the library can called by the example code below where `flash` can be replaced by a constructor of the user's choice.
- The constructor is used to call a function from the SPIFlash library.

### 5.2.3 Example code:

[0] learncpp.com.

```
#include <SPIMemory.h>

SPIFlash flash;   //This is the constructor. This example uses 'flash' as the␣
↪constructor

  void setup() {
  ...
  }

  void loop() {
  flash.readByte(...);  //The constructor 'flash' is used to call the function
↪'readByte()' from the library
  ...
  }
```

### 5.2.4 Related Errors `None`

### 5.2.5 Advanced Use

#### Defining a custom Chip Select pin

The library can also called by declaring the `cs` parameter in the constructor where `cs` is the uuser defined Chip Select pin for the flash module.

```
#include <SPIMemory.h>

SPIFlash flash(33);   // The library uses the `pin 33` as the Chip Select pin instead␣
↪of the default

void setup() {
  ...
}

void loop {
  ...
}
```

---

#### Using a non-default SPI interface†[0]

†

- **The library currently only supports using non-default SPI interfaces on the following architectures:**
  - SAMD
  - STM32
- The csPin **MUST** be declared if using a non-default SPI interface.
- Only available if library > v3.0.0

[0] This is currently only supported on the SAMD and STM32 architectures.

```
#include <SPIMemory.h>

SPIFlash flash(33, &SPI1);  // The library now uses the 'SPI1' interface instead of the default 'SPI0'.
                            // It also uses pin 33 instead of the default Chip Select pin
void setup() {
  ...
}

void loop {
  ...
}
```

## 5.3 Library Instantiation

These functions set up the library for use and should be called as required.

### 5.3.1 begin() `Mandatory`

bool **begin** (uint32_t *flashChipSize* = 0)

#### Parameters `Optional`

uint32_t **flashChipSize**
Refer to *Using with non-supported flash memory*

#### What it does

#### Returns `boolean`

The function returns `TRUE` if successfully executed and `FALSE` if otherwise.

- Must be called at the start in `void setup()`. This function detects the type of chip being used and sets parameters accordingly.
- This function is essential to the functioning of the library and must be called before any other calls are made to the library.

#### Example code

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin();  // This function has to be called first - before any other functions
                  // from this library are called
```

(continues on next page)

(continued from previous page)

```
  ...
}

void loop() {
  ...
}
```

### Related Errors `CALLBEGIN UNKNOWNCAP UNKNOWNCHIP`

- If this function is not called the library throws the error - `CALLBEGIN`.
- If the *chip's capacity* cannot be identified the library throws the error - `UNKNOWNCAP`.
- If the *chip cannot be ID'd* the library throws the error - `UNKNOWNCHIP`.

### Advanced use `flashChipSize`

### Using with non-supported flash memory

- An optional `flashChipSize` parameter can be declared as an argument with this function (if library version > v2.6.0)
- In an instance where the library is being used with a flash memory chip that is not officially supported by the Library, declaring the chip storage size in *bytes* as the `flashChipSize` parameter can - in many instances - enable the library to work with the chip

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin(MB(32));   // This sets the flash chip size to 32 Megabytes
                         // - Refer to defines.h for the expansion of the MB(32) macro
  ...
}

void loop() {
  ...
}
```

## 5.3.2 setClock() `Advanced use only: Use with care`

void **setClock**(uint32_t *clockSpeed*)

### Parameters `Mandatory`

uint32_t **clockSpeed**
A 32 bit unsigned integer that represents SPI Clock Speed in Hertz

### What it does

### Returns `Nothing`

- This is an optional function and is used to set the SPI clock speed for all further comms using the library.
- If required, this function must be called straight after begin().
- This function takes a 32-bit value (in Hertz) as replacement for the default maximum clock speed (104MHz for Winbond NOR flash) thereby initiating future SPI transactions with the user-defined clock speed.

### Example code

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin();
  flash.setClock(20000000);     // this sets the clock spped to 20,000,000 Hz - i.e. 20MHz
  ...
}

void loop() {
  ...
}
```

### Related Errors `None`

### Advanced use `N/A`

## 5.4 Chip ID

`SFDP compatibility dependent`

A number of functions are available to the library user to identify chip in use.

**Note:** The library is designed to identify the flash memory chip in use when the `begin();` function is called (using the internal function `_chipID();`). This happens in two ways:

- If the flash memory supports the SFDP standard, then, the SFDP tables are used to identify the chip. The information from the tables is usually enough to let the library execute all functions without impediment, so any Chip ID errors thrown (if any) can be and, are ignored.
- If the flash memory does not support the SFDP standard, three situations can arise:
  - If the chip is officially supported by the library, its can be identified by internal methods.
  - If the chip is not officially supported, then the user has to declare the size as an argument when calling the `begin();` function. This information is usually enough to let the library execute all functions without impediment, so any Chip ID errors thrown (if any) can be and, are ignored.
  - If the chip canot be ID'd and the user does not declare a size in `begin();`, the library throws an error - `UNKNOWNCHIP` as soon as the function `begin();` is called. (Refer to Diagnostics & Error reporting)

### 5.4.1 getManID()

uint16_t **getManID** (void)

**Parameters** `None`

**What it does**

**Returns** `uint16_t`

Returns the Manufacturer ID as a 16 bit unsigned integer

**Example code:**

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin();
  uint16_t manID = flash.getManID();       // Function is used to get the manufacturer␣
↪ID and store it as
                                           // a 16 bit unsigned integer
  Serial.print(F("Chip Manufacturer ID: 0x"));
  Serial.println(manID, HEX);                   // The manufacturer ID is printed as a␣
↪Hexadecimal number
  ...
}

void loop() {
  ...
}
```

**Related Errors** `None`

**Advanced use** `N/A`

### 5.4.2 getJEDECID()

uint32_t **getJEDECID** (void)

#### Parameters `None`

#### What it does

#### Returns `uint32_t`

Returns the JEDEC ID as a 32 bit unsigned integer

#### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin();
  uint32_t JEDEC = flash.getJEDECID();       // Function is used to get the JEDEC ID␣
↪and store it as
                                             // a 32 bit unsigned integer
  Serial.print(F("JEDEC ID: 0x"));
  Serial.println(JEDEC, HEX);                   // The JEDEC ID is printed as a␣
↪Hexadecimal number
  ...
}

void loop() {
  ...
}
```

#### Related Errors `UNKNOWNCHIP`

The way this function executes depends on whether the flash memory chip complies with the SFDP standard.

- If the chip supports SFDP (immaterial of official support), then, the library will work as it should - immaterial of whether or not it can read the JEDEC ID (even if it throws the error `UNKNOWNCHIP`).
- If the chip is does not support SFDP and the chip is unable to read the JEDEC ID (internally in the `begin();` function), then it throws the error `UNKNOWNCHIP`

#### Advanced use `N/A`

---

### 5.4.3 getUniqueID() `Memory IC dependent`

uint64_t **getUniqueID**(void)

#### Parameters `None`

#### What it does

### Returns `uint64_t`

Returns the flash memory chip's unique ID as a 64 bit unsigned integer

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin();
  uint64_t uniqueID = flash.getUniqueID();       // Function is used to get the unique␣
↪ID and store it as
                                                 // a 64 bit unsigned integer
  Serial.print(F("Unique ID: 0x"));
  Serial.println(uniqueID, HEX);                   // The unique ID is printed as a␣
↪Hexadecimal number
  ...
}

void loop() {
  ...
}
```

### Related Errors `None`

### Advanced use `N/A`

---

## 5.4.4 getCapacity()

uint32_t **getCapacity**(void)

### Parameters `None`

### What it does

### Returns `uint32_t`

Returns the flash memory chip's capacity as a 32 bit unsigned integer

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
```

(continues on next page)

(continued from previous page)

```
  flash.begin();
  uint32_t cap = flash.getCapacity();      // Function is used to get the unique ID␣
↪and store it as
                                               // a 32 bit unsigned integer
  Serial.print(F("Capacity: "));
  Serial.println(cap);               // The unique ID is printed as a decimal number -
↪ in bytes
  ...
}

void loop() {
  ...
}
```

### Related Errors `UNKNOWNCAP`

If the chip's capacity cannot be determined, the library throws an error - `UNKNOWNCAP` as soon as the function `begin();` is called. (Refer to Diagnostics & Error reporting)

### Advanced use `N/A`

**Note:** The way this function executes depends on whether the flash memory chip complies with the SFDP standard.

- **The chip's capacity is determined in one of three ways:**
  - If the chip supports SFDP (immaterial of official support), then, the chip's capacity is read from the SFDP tables.
  - If the chip is officially supported by the library, its capacity is already known.
  - If the chip is not officially supported, then the user has to declare the size as an argument when calling the `begin()` function.

## 5.4.5 getMaxPage()

uint32_t **`getMaxPage`**(void)

### Parameters `None`

### What it does

### Returns `uint32_t`

Returns the number of physical *pages* in the flash memory as a 32 bit unsigned integer

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

void setup() {
  flash.begin();
  uint32_t maxPage = flash.getMaxPage();   // Function is used to get the number of␣
↪pages and store it as
                                           // a 32 bit unsigned integer
  Serial.print(F("Maximum pages: "));
  Serial.println(maxPage);                   // The number of pages is printed
  ...
}

void loop() {
  ...
}
```

### Related Errors `None`

### Advanced use `N/A`

**Note:** The way this function executes depends on whether the flash memory chip complies with the SFDP standard.

- If the chip supports SFDP, then, the chip's capacity and page size (in bytes) are read from the SFDP tables.
- If the chip does not support SFDP, then the chip's capacity is determined in one of two ways (refer to the note in *getCapacity()*). The pagesize uses the default (and most common) value of 256 bytes per page.

## 5.5 Read functions

These functions enable the user to read data that is stored on the flash memory chip. The various functions listed return different data types and can be used as required.

There are two basic types of read functions - one that is independent of data type being read, the other is dependent on the data type. Please refer to the links below for further details.

### 5.5.1 Data type-independent read function

bool **`readAnything`**(uint32_t *_addr*, T &*data*, bool *fastRead* = false)

### Parameters `Mandatory & Optional`

uint32_t **_addr**

Address in memory where the data is to be read from. `Mandatory`

T &**data**

Variable to save the data to. `Mandatory`

bool **fastRead**

Refer to *Advanced use* `Optional`

### What it does

Reads any type of variable / struct (any sized value) (starting) from a specific address and saves it to the variable `T& data` provided as an argument.

### Returns `boolean`

Returns `TRUE` if successful, `FALSE` if unsuccessful

**Note:** This function can be used to replace any of the other read functions (except `readByteArray()` and `readCharArray()`). However, if used for anything other than structs, this function runs slower than the data type-specific ones.

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

struct testStruct {
  uint8_t _byte = 8;
  uint16_t _int = 269;
  uint32_t _long = 99432;
  float _float = 3.14;
  String _str = "This is a test string";
  uint8_t _array[8] = {0,1,2,3,4,5,6,7};
} dataIn;
uint32_t _address;

void setup() {
  flash.begin();
  _address = flash.getAddress(sizeof(dataIn));
  Serial.print(F("Address = "));
  Serial.println(_address);
  Serial.print(F("readAnything()"));
  if (!flash.readAnything(_address, dataIn)) { // Function is used to get the data␣
↪from
                                              // address '_address' and save it to the␣
↪struct 'test'
```

(continues on next page)

(continued from previous page)

```
    Serial.println(F("Failed"));
  }
  else {
    Serial.println(F("Passed"));
  }
  ...
}

void loop() {
  ...
}
```

### 5.5.2 Data type-dependent read functions

These functions are designed to read a specific data type from the flash memory. There are two types of data type-dependent read functions as listed below

#### Single variable read functions

These functions are designed to read a single variable - of a specific data type - from the flash memory. Each supported type of variable has a function specific to it - as listed below:

uint8_t **readByte** (uint32_t *_addr*, bool *fastRead* = false)

- Reads an unsigned 8 bit integer (a `byte`) from the address specified, and returns it.

int8_t **readChar** (uint32_t *_addr*, bool *fastRead* = false)

- Reads a signed 8 bit integer (a `char`) from the address specified, and returns it.

int16_t **readShort** (uint32_t *_addr*, bool *fastRead* = false)

- Reads a signed 16 bit integer (a `short`) from the address specified, and returns it.

uint16_t **readWord** (uint32_t *_addr*, bool *fastRead* = false)

- Reads an unsigned 16 bit integer (a `word`) from the address specified, and returns it.

int32_t **readLong** (uint32_t *_addr*, bool *fastRead* = false)

- Reads a signed 32 bit integer (a `long`) from the address specified, and returns it.

uint32_t **readULong** (uint32_t *_addr*, bool *fastRead* = false)

- Reads an unsigned 32 bit integer (an `unsigned long`) from the address specified, and returns it.

float **readFloat** (uint32_t *_addr*, bool *fastRead* = false)

- Reads a `float` from the address specified, and returns it.

bool **readStr** (uint32_t *_addr*, String *&data*, bool *fastRead* = false)

- Reads a `String` from the address specified, and returns it.

#### Parameters `Mandatory & Optional`

uint32_t **_addr**

Address in memory where the data is to be read from. `Mandatory` bool **fastRead**

Refer to *Advanced use* `Optional`

### What they do

Return the value (of the datatype specified) that is stored at the address provided

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

uint8_t dataIn;     // This data type should be changed depending on the type of data
                      // being read from the flash memory
uint32_t _address;

void setup() {
  flash.begin();
  _address = flash.getAddress(sizeof(dataIn));
  dataIn = flash.readByte(_address);     // This function should be changed depending␣
↪on the type of data
                                         // being read from the flash memory
  Serial.print(F("Address = "));
  Serial.println(_address);
  Serial.print(F("Data read : 0x"));
  Serial.println(dataIn, HEX);
}

void loop() {
}
```

### Array read functions

bool **readByteArray**(uint32_t *_addr*, uint8_t **data_buffer*, size_t *bufferSize*, bool *fastRead* = false)

- Reads an `array of bytes` from the address specified, and saves the values to the `data_buffer` array provided.

bool **readCharArray**(uint32_t *_addr*, char **data_buffer*, size_t *buffer_size*, bool *fastRead* = false)

- Reads an `array of chars` from the address specified, and saves the values to the `data_buffer` array provided.

### Parameters `Mandatory & Optional`

uint32_t **_addr**

Address in memory where the data is to be read from. `Mandatory`

uint8_t ***data_buffer**

Pointer to data buffer to write the array to. `Mandatory` size_t **buffer_size**

Size of the array to be read out from flash memory. `Mandatory`

bool **fastRead**

Refer to *Advanced use* `Optional`

### What they do

Return the array of values of the datatype and size specified by the user, that is stored (starting) at the address provided

### Returns `boolean`

Returns `TRUE` if data read successfully, else returns `FALSE`

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

#define _bufferSize 8

uint8_t dataIn[_bufferSize];
// This data type should be changed depending on the type of data being read from the␣
↪flash memory
uint32_t _address;

void setup() {
  flash.begin();
  _address = flash.getAddress(sizeof(dataIn));
  Serial.print(F("Address = "));
  Serial.println(_address);

  dataIn = flash.readByteArray(_address, dataIn, _bufferSize);
  // This function should be changed depending on the type of data being read from␣
↪the flash memory

  Serial.print(F("Data read: "));
  for (uint8_t i = 0; i < _bufferSize; i++) {
    Serial.print(dataIn[i]);
    Serial.print(F(", "));
  }
  Serial.println();

}

void loop() {
}
```

### 5.5.3 Related Errors CHIPISPOWEREDDOWN CALLBEGIN OUTOFBOUNDS CHIPBUSY

- If the chip has previously been powered down and hasn't been powered up prior to calling this function, the library throws the error `CHIPISPOWEREDDOWN`
- If `begin()` has not been called in `void setup()`, the library throws the error `CALLBEGIN`
- If the address to be read from is out of bounds - i.e. greater than the available memory on the chip - and address overflow has been disabled, the library throws the error `OUTOFBOUNDS`
- If the chip is busy executing the a command passed to it previously or is locked up, the library throws the error `CHIPBUSY`

### 5.5.4 Advanced use fastRead

- All read commands take a last boolean argument `fastRead`. This argument defaults to `FALSE`, and does not need to be specified when calling a function.

For example:

```
...

//Calling
flash.readByte(addressToReadFrom);
//or
flash.readByte(addressToReadFrom, FALSE);
//yields the same results.

...
```

- However, when this argument is set to TRUE, it carries out the Fast Read instruction so data can be read at up to the memory's maximum frequency.

```
...

//Calling
flash.readByteArray(addressToReadFrom, bufferToReadTo, bufferSize, TRUE);
//instead of
flash.readByteArray(addressToReadFrom, bufferToReadTo, bufferSize);
// will result in faster read speeds for very large data arrays.

...
```

**This is useful only when reading very large amounts of data from the flash memory. If used for small arrays or individual variables, it will slow down the read function.**

## 5.6 Write functions

These functions enable the user to write data that is stored on the flash memory chip. The various functions listed write different data types and can be used as required.

There are two basic types of write functions - one that is independent of data type being write, the other is dependent on the data type. Please refer to the links below for further details.

### 5.6.1 Data type-independent write function

bool **writeAnything** (uint32_t *_addr*, **const** T &*data*, bool *errorCheck* = true)

#### Parameters `Mandatory & Optional`

uint32_t **_addr**

Address in memory where the data is to be written to. `Mandatory`

T &**data**

Variable to write. `Mandatory`

bool **errorCheck**

Refer to *Advanced use* `Optional`

#### What it does

Writes any type of variable / struct (any sized value) (starting) from a specific address (user provided)

#### Returns `boolean`

Returns `TRUE` if successful, `FALSE` if unsuccessful

**Note:** This function can be used to replace any of the other write functions (except `writeByteArray()` and `writeCharArray()`). However, if used for anything other than structs, this function runs slower than the data type-specific ones.

#### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

struct testStruct {
  uint8_t _byte = 8;
  uint16_t _int = 269;
  uint32_t _long = 99432;
  float _float = 3.14;
  String _str = "This is a test string";
  uint8_t _array[8] = {0,1,2,3,4,5,6,7};
} dataOut;
uint32_t _address;

void setup() {
  flash.begin();
  _address = flash.getAddress(sizeof(dataIn));
  Serial.print(F("Address = "));
  Serial.println(_address);
```

(continues on next page)

(continued from previous page)

```
  Serial.print(F("writeAnything()"));
  if (!flash.writeAnything(_address, dataOut)) { // Function is used to write the␣
↪data to
                                                // address '_address'
    Serial.println(F("Failed"));
  }
  else {
    Serial.println(F("Passed"));
  }
  ...
}

void loop() {
  ...
}
```

### 5.6.2 Data type-dependent write functions

These functions are designed to write a specific data type to the flash memory. There are two types of data type-dependent write functions as listed below

#### Single variable write functions

These functions are designed to write a single variable - of a specific data type - to the flash memory. Each supported data type has a function specific to it - as listed below:

bool **writeByte** (uint32_t *_addr*, uint8_t *data*, bool *errorCheck* = true)

- Writes an unsigned 8 bit integer - a `byte`- to the address specified.

bool **writeChar** (uint32_t *_addr*, int8_t *data*, bool *errorCheck* = true)

- Writes a signed 8 bit integer - a `char`- to the address specified.

bool **writeShort** (uint32_t *_addr*, int16_t *data*, bool *errorCheck* = true)

- Writes a signed 16 bit integer - a `short`- to the address specified.

bool **writeWord** (uint32_t *_addr*, uint16_t *data*, bool *errorCheck* = true)

- Writes an unsigned 16 bit integer - a `word`- to the address specified.

bool **writeLong** (uint32_t *_addr*, int32_t *data*, bool *errorCheck* = true)

- Writes a signed 32 bit integer - a `long`- to the address specified.

bool **writeULong** (uint32_t *_addr*, uint32_t *data*, bool *errorCheck* = true)

- Writes an unsigned 32 bit integer - an `unsigned long`- to the address specified.

bool **writeFloat** (uint32_t *_addr*, float *data*, bool *errorCheck* = true)

- Writes a `float` to the address specified.

bool **writeStr** (uint32_t *_addr*, String *&data*, bool *errorCheck* = true)

- Writes a `String` to the address specified.

### Parameters Mandatory & Optional

uint32_t **_addr**

Address in memory where the data is to be written to. Mandatory

size_t **data**

Data variable to write to the flash memory. Mandatory

bool **errorCheck**

Refer to *Advanced use* Optional

### What they do

Write the data (of the datatype and size specified by the user) to the address provided

### Returns boolean

Returns TRUE if data written successfully, else returns FALSE

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

uint32_t _address;
// This data type should be changed depending on the type of data being write to the␣
↪flash memory
String dataOut = "This is a test String!";


void setup() {
  flash.begin();
  _address = flash.getAddress(sizeof(dataIn));
  Serial.print(F("Address = "));
  Serial.println(_address);

  Serial.print(F("Data write "));
  // This function should be changed depending on the type of data being written to␣
↪the flash memory
  if (flash.writeStr(_address, dataOut)) {
    Serial.println(F("successful"));
  }
  else {
    Serial.println(F("failed"));
  }
}

void loop() {
}
```

### Array write functions

bool **writeByteArray**(uint32_t *_addr*, uint8_t **data_buffer*, size_t *bufferSize*, bool *errorCheck* = true)

- Writes an `array of bytes` to the address specified.

bool **writeCharArray**(uint32_t *_addr*, char **data_buffer*, size_t *bufferSize*, bool *errorCheck* = true)

- Writes an `array of chars` to the address specified.

### Parameters `Mandatory & Optional`

uint32_t **_addr**

Address in memory where the data is to be written to. `Mandatory`

uint8_t ***data_buffer**

Pointer to data buffer to write the array from. `Mandatory`

size_t **bufferSize**

Size of the array to be written out to flash memory. `Mandatory`

bool **errorCheck**

Refer to *Advanced use* `Optional`

### What they do

Write the array of values of the datatype and size specified by the user, to the address provided

### Returns `boolean`

Returns `TRUE` if data written successfully, else returns `FALSE`

### Example code:

```
#include <SPIMemory.h>

SPIFlash flash;

#define _bufferSize 8

uint8_t dataOut[_bufferSize] = {0,1,2,3,4,5,6,7};
// This data type should be changed depending on the type of data being written to␣
↪the flash memory
uint32_t _address;

void setup() {
  flash.begin();
  _address = flash.getAddress(sizeof(dataIn));
  Serial.print(F("Address = "));
  Serial.println(_address);

```

(continues on next page)

(continued from previous page)

```
  Serial.print(F("Data write: "));
  if (flash.writeByteArray(_address, dataOut, _bufferSize) {
  // This function should be changed depending on the type of data being written to␣
↪the flash memory
    Serial.println(F("Successful"));
  }
  else {
    Serial.println(F("Failed"));
  }

}

void loop() {
}
```

### 5.6.3 Related Errors `CHIPISPOWEREDDOWN` `CALLBEGIN` `OUTOFBOUNDS` `CHIPBUSY` `PREVWRITTEN`

- If the chip has previously been powered down and hasn't been powered up prior to calling this function, the library throws the error `CHIPISPOWEREDDOWN`
- If `begin()` has not been called in `void setup()`, the library throws the error `CALLBEGIN`
- If the address to be read from is out of bounds - i.e. greater than the available memory on the chip - and address overflow has been disabled, the library throws the error `OUTOFBOUNDS`
- If the chip is busy executing the a command passed to it previously or is locked up, the library throws the error `CHIPBUSY`
- If the address to be written to already contains data (i.e. has not been erased), the library throws the error `PREVWRITTEN`.*[0]

#### footnotes

### 5.6.4 Advanced use `errorCheck` `HIGHSPEED`

---

#### errorCheck `WARNING: Data corruption likely`

- All write functions have Error checking turned on by default - i.e. every byte written to the flash memory will be checked against the data provided by the user.
- This is controlled by an optional boolean argument `errorCheck`. This argument defaults to `TRUE`, and does not need to be specified when calling a function.

For example:

```
...

//Calling
flash.writeByte(addressToWriteTo, byteOfData);
//or
```

(continues on next page)

[0] Refer to *Note on HIGHSPEED mode*

(continued from previous page)

```
flash.writeByte(addressToWriteTo, byteOfData, FALSE);
//yields the same results.

...
```

- Users who require greater write speeds can disable this function by setting the `errorCheck` argument in any write function to `NOERRCHK`

For example:

```
...

//Calling
flash.writeByte(addressToWriteTo, byteOfData, NOERRCHK);
//instead of
flash.writeByte(addressToWriteTo, byteOfData);
// will result in faster write speeds at the risk of not catching any errors in the writing process.

...
```

**This is useful only when the program calls for increased writing speed over data integrity. Data writes cannot be guaranteed free of errors if errorCheck is turned off**

### Highspeed mode `WARNING: Data corruption likely`

The library - by default - checks to see if the address being written to contains pre-existing data. If it finds pre-existing data, it throws the error `PREVWRITTEN`. This prevents the write function from running as fast as it theoretically could.

If the user requires the library to operate at the highest speed possible, they must be sure that the user code does the following before every data write:

- Makes sure that the flash memory chip does not contain any data before using it.
- Erases a sector before writing to it.
- Keeps track of addressing and makes sure no two bytes of data are ever written to the same address.
- Tracks previously written addresses in the user code and makes sure to erase them before writing new data at those addresses.

Then, if `#define HIGHSPEED` is uncommented in SPIFlash.h before the user code is compiled and uploaded, the write functions will run as fast as the flash memory hardware will permit them to.

**Note:** `**WARNING**`

Please note that using a combination of the methods listed in *High speed mode* and *Error checking* will result in the highest possible write speed from the library. However, this will result in the highest probability of write errors / data corruption as well.

`**WARNING**`

# CHAPTER 6

# Contribute

## 6.1 Issue Tracker

Raise any feature requests or report any bugs at the link above. Please use the template that is presented when you raise an issue.

# CHAPTER 7

# License

## 7.1 GNU GENERAL PUBLIC LICENSE

*Version 3, 29 June 2007*

### 7.1.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

### 7.1.2 TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

{one line to give the program's name and a brief idea of what it does.} Copyright (C) {year} {name of author}

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

{project} Copyright (C) {year} {fullname} This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

CHAPTER 8

# Index

CHAPTER 9

# Status & stats

# Index