

# SRAM Dynamic Memory Allocation

<b>1. Introduction .....</b>	<b>2</b>
<b>2. Module Features.....</b>	<b>2</b>
<b>3. List of Component Modules.....</b>	<b>3</b>
<b>4. Using the Library Module in a Project.....</b>	<b>3</b>
<b>5. List of Shared Parameters .....</b>	<b>4</b>
<i>Shared Data Bytes .....</i>	<i>4</i>
<i>Shared Functions .....</i>	<i>4</i>
<i>Shared Macros.....</i>	<i>4</i>
<b>6. Functions.....</b>	<b>5</b>
<b>7. Macros .....</b>	<b>6</b>
<b>8. Error and Status Flags.....</b>	<b>7</b>

## 1. Introduction

This module adds simple dynamic memory allocation within SRAM for PIC18 devices. The memory allocation scheme leans towards a 'KISS' design methodology, which allows for very efficient allocation and de-allocation while maintaining a small Program Memory footprint. As expected though, this design also imposes some limitations.

How it works:

The model is based on a simple form of a linked list. A block of memory referred to as the dynamic heap is split into segments. Each segment has a single byte header that references the next segment in the list as well as indicating whether the segment is allocated. Seven bits are used to reference the next segment, which consequently the reference implicitly identifies the length of the segment, and one bit is used to determine if it is allocated. Thus the maximum allocable block is 126 bytes or less, depending on whether the heap is less than 126 bytes or not. The maximum number of allocations is limited only by the defined heap size.

Recommendations:

Although this model will allow dynamic allocation down to a single byte, doing so sacrifices performance. With more segments within the heap, more time is required to attempt to allocate memory. Plus every segment requires a header byte; therefore, smaller segments require more memory. There is also the possibility of fragmentation, which could ultimately doom an application by reducing the largest allocable block of memory. Thus the recommendation is to allocate at least 8 bytes of memory.

Linker Script Setup:

This module requires a memory block to be defined in the linker script for the dynamic memory heap. The following is an example, note the section name must be `_SRAM_ALLOC_HEAP`:

```
SECTION      NAME=_SRAM_ALLOC_HEAP  RAM=sramalloc
```

Refer to the script provided for a complete example.

## 2. Module Features

The SRALLOC module has the following features:

- Small code size.
- Efficient allocation and de-allocation.
- Configurable heap size.
- Functions like the standard *malloc* and *free*.

### 3. List of Component Modules

<code>Sralloc.zip</code>	This zip contains an example usage file, project file, and linker script file.
<code>Sralloc.c</code>	This is the core code implementing the dynamic memory allocation.
<code>Sralloc.h</code>	Header for dynamic memory allocation. Include this in every file that uses memory allocation.

### 4. Using the Library Module in a Project

Please follow below steps to use this library module in your project.

1. Use the Application Maestro to configure your code as required.
2. At the Generate Files step, save the output to the directory where your code project resides.
3. Launch MPLAB, and open the project's workspace.
4. Verify that the Microchip language tool suite is selected (*Project>Select Language Toolsuite*).
5. In the Workspace view, right-click on the "Source Files" node. Select the "Add Files" option. Select `sralloc.c` and click **OK**.
6. Now right-click on the "Linker Scripts" node and select "Add Files". Add the appropriate linker file (`.lkr`) for the project's target microcontroller. The script will need to be modified.
7. Add any other files that the project may require. Save and close the project.
8. Add the header `sralloc.h` to any file that requires usage of dynamic memory allocation.
9. To use the module in your application, invoke the functions as needed.

## 5. List of Shared Parameters

### ***Shared Data Bytes***

NA

### ***Shared Functions***

SRAMInitHeap

This function initializes the dynamic heap specified in the linker script. The heap must be initialized before use.

SRAMalloc

This function attempts to allocate a block of memory within the heap. The function returns a pointer to the first byte or NULL if not successful.

SRAMfree

This function frees a block of memory that was previously allocated.

### ***Shared Macros***

NA

## 6. Functions

Function	SRAMInitHeap
Preconditions	None
Overview	This function initializes the dynamic heap. It inserts segment headers to maximize segment space. This function must be called at least one time. And it could be called more times to reset the heap.
Input	None
Output	None
Side Effects	None
Stack Requirement	1 level deep

Function	SRAMalloc
Preconditions	A memory block must be allocated in the linker, and the memory headers and tail must already be set via the function RAMInitHeap().
Overview	This function allocates a chunk of memory from the heap. The maximum segment size for this version is 126 bytes. If the heap does not have an available segment of sufficient size it will attempt to create a segment; otherwise a NULL pointer is returned. If allocation is successful then a pointer to the requested block is returned. The calling function must maintain the pointer to correctly free memory at runtime.
Input	unsigned char nBytes - Number of bytes to allocate.
Output	unsigned char * - A pointer to the requested block of memory.
Side Effects	None
Stack Requirement	2 levels deep

Function	SRAMfree
Preconditions	The pointer must have been returned from a previously allocation via SRAMalloc().
Overview	This function de-allocates a previously allocated segment of memory. The pointer must be a valid pointer returned from SRAMalloc(); otherwise, the segment may not be successfully de-allocated, and the heap may be corrupted.
Input	unsigned char * pSRAM - pointer to the allocated memory
Output	none
Side Effects	none
Stack Requirement	1 level deep

## 7. Macros

There are no macros in the module.

## **8. Error and Status Flags**

There are no public flags in this module.