

N I N T E N D O

NITRO-System

Memory Manager

Three Heaps Specialized for Games

Version 1.0.9a

**The contents in this document are highly
confidential and should be handled accordingly.**

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo of America Inc. and/or Nintendo Company Ltd. and are protected by Federal copyright law. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

1	Preface.....	7
2	Extended Heap Manager.....	8
2.1	Creating Heaps	8
2.2	Allocating Memory Blocks.....	8
2.2.1	Allocating and Freeing Memory Blocks.....	8
2.2.2	The Minimum Memory Block Allocation Unit.....	8
2.2.3	Memory Allocation Procedure	9
2.2.4	Allocating Memory Blocks from the Back of the Heap Region.....	9
2.3	Specifying Alignment.....	11
2.4	Changing Memory Block Size.....	11
2.5	Acquiring Free Space.....	12
2.6	Group ID.....	12
2.7	Processing Memory Blocks.....	13
2.8	Acquiring Memory Block Information	13
2.9	Checking Heaps and Memory Blocks.....	13
3	Frame Heap Manager.....	14
3.1	Creating Heaps	14
3.2	Allocating Memory Blocks.....	14
3.2.1	Allocating and Freeing Memory Blocks.....	14
3.2.2	The Minimum Memory Block Allocation Unit.....	15
3.3	Specifying Alignment.....	15
3.4	Freeing Memory Blocks	15
3.5	Storing and Returning to a Memory Block Allocation Status.....	16
3.6	Adjusting Heap Region Size	17
3.7	Changing the Size of Memory Blocks.....	18
3.8	Acquiring the Size That Can Be Allocated	18
4	Unit Heaps	19
4.1	Creating Heaps	19
4.2	Memory Block Allocation.....	20
4.2.1	Allocating and Freeing Memory Blocks.....	20
4.2.2	The Minimum Memory Block Allocation Unit.....	20
4.3	Specifying Alignment.....	21
4.4	Acquiring the Number of Memory Blocks That Can Be Allocated	21
5	Functionality Common to Each Heap	22
5.1	Heap Options	22
5.2	Changing the Values to Fill When Debugging	22
5.3	Displaying Heap Contents	23

5.4	Acquiring Heap Regions	23
6	Multi-Heap Management	24
6.1	Multi-Heaps	24
6.2	Freeing Multi-Heap Memory	24
6.3	Managing Heaps with a Tree.....	24

Tables

Table 2-1 Functions for Creating and Destroying Extended Heaps	8
Table 2-2 Functions for Allocating and Freeing Memory Blocks	8
Table 2-3 Memory Block Allocation Modes	9
Table 2-4 Allocation Mode Setting and Acquisition.....	9
Table 2-5 Allocation Modes Used by the Functions	9
Table 2-6 Function for Changing Memory Block Size	11
Table 2-7 Functions For Acquiring Free Space, Etc.	12
Table 2-8 Functions for Setting and Acquiring Group IDs	12
Table 2-9 Functions for Acquiring Memory Block Information.....	13
Table 2-10 Functions that check extended heaps and memory blocks	13
Table 3-1 Functions for Creating and Destroying Heaps	14
Table 3-2 Functions for Allocating Memory Blocks.....	14
Table 3-3 Methods Of Freeing Memory Blocks	15
Table 3-4 Function for Freeing Memory Blocks.....	15
Table 3-5 Values Specified in the Function for Freeing Memory Blocks	16
Table 3-6 Functions for Storing a Memory Block Allocation Status and Returning To It.....	17
Table 3-7 Function for Reducing Heap Region Size	17
Table 3-8 Function for changing the size of memory block.....	18
Table 3-9 Functions for Obtaining the Size That Can Be Allocated	18
Table 4-1 Functions for Creating and Destroying Heaps	19
Table 4-2 Function for Allocating Memory Blocks	20
Table 4-3 Function for Acquiring the Number of Memory Blocks That Can Be Allocated	21
Table 5-1 Options That Can Be Specified When Creating a Heap	22
Table 5-2 Functions for Setting and Acquiring Values That Are Filled At Debug Time.....	22
Table 5-3 Type of heap operation to fill.	23
Table 5-4 Function for Displaying Internal Heap Information	23
Table 5-5 Functions For Acquiring Heap Regions.....	23
Table 6-1 Function for Searching for the Heap That Allocated a Memory Block	24

Figures

Figure 2-1 Procedure for Allocating Extended Heap Memory Blocks	9
Figure 2-2 Mechanism That Fragments Memory Blocks	10
Figure 2-3 Measures for Dealing with Extended Heap Memory Block Fragmentation.....	10
Figure 2-4 Changing Extended Heap Memory Block Size	11
Figure 3-1 Frame Heap Memory Allocation	14
Figure 3-2 Mechanism for Storing and Returning to Frame Heap Memory Block Allocation Status.....	16
Figure 3-3 Adjusting Frame Heap Size.....	17
Figure 3-4 Adjusting the size of the frame heap memory block.....	18
Figure 4-1 Unit Heap Memory Allocation	20

Revision History

Version	Revision Date	Description
1.0.9a	2007/04/27	<ul style="list-style-type: none"> Corrected typographical errors. Changed Revision History dates to international format.
1.0.9	2005/01/05	<ul style="list-style-type: none"> Changed instances of "NITRO" to "Nintendo DS."
1.0.8	2004/08/20	<ul style="list-style-type: none"> Added "Checking Heaps and Memory Blocks" to the Extended Heap
1.0.7	2004/08/02	<ul style="list-style-type: none"> Added "Changing Size of Memory Blocks" to Frame Heap
1.0.6	2004/06/10	<ul style="list-style-type: none"> Revised description of "Heap Options."
1.0.5	2004/04/12	<ul style="list-style-type: none"> Revised Misspelling
1.0.4	2004/03/30	<ul style="list-style-type: none"> Changed title and header. Added explanation to all areas that were difficult to understand. Corrected spelling errors and omissions.
1.0.3	2004/03/29	<ul style="list-style-type: none"> Corrected spelling errors and omissions. Deleted 2 sections from the main body of the Introduction Revised the descriptions of the functionality of each heap. Added "Functionality Common To Each Heap".
1.0.2	2004/03/25	<ul style="list-style-type: none"> Added description of Extended Heap API. Added to Extended Heap "Creating Heaps". Revised Extended Heap "Memory Allocation Procedure". Revised Extended Heap "Specifying Alignment". Deleted Extended Heap "Free Special Memory". Added to Extended Heap "Acquiring Free Capacity". Changed values in Extended Heap "Group ID". Added to Extended Heap "Acquiring Memory Block Information". Added to Frame Heap "Creating Heaps". Revised Frame Heap "Specifying Alignment". Added to Frame Heap "Acquiring the Size That Can Be Allocated". Added to Unit Heap "Creating Heaps". Revised Unit Heap "Memory Block Allocation". Added to Unit Heap "Minimum Unit When Allocating Memory Blocks". Added to Unit Heap "Specifying Alignment". Added to Unit Heap "Acquiring the Number of Memory Blocks That Can Be Allocated".
1.0.1	2004/02/06	<ul style="list-style-type: none"> Deleted Extended Heap "Freeing by Group ID". Added to Extended Heap "Processing Memory Blocks". Changed Unit Heap algorithm.

1 Introduction

The Nintendo DS has 4 megabytes of main memory. It is difficult to manage this much memory with a memory map. A memory manager allows you to dynamically allocate and free memory, eliminating the need for memory maps or other such tools.

On the other hand, the Nintendo DS memory size is relatively small compared to a PC or workstation. There are circumstances that are specific to a game machine in which the use of the generic `malloc()` and `free()` functions are not sufficient. Nitro-System provides three memory managers that are created specifically for use with the NITRO-System. These memory managers have additional functionality beyond the heap mechanisms commonly used for games.

2 Extended Heap Manager

The Extended Heap Manager can allocate and free memory, in much the same way as the `malloc()` function and the `free()` function in the C standard library. In addition to the basic functionality of allocating and freeing memory, this manager has additional features for game software. This section gives an overview of the Extended Heap Manager.

2.1 Creating Heaps

In order to use the Extended Heap Manager, you must first create an extended heap. The following functions create and destroy (delete) extended heaps.

Table 2-1 Functions for Creating and Destroying Extended Heaps

Function	Description
<code>NNS_FndCreateExpHeap()</code>	Creates an extended heap.
<code>NNS_FndCreateExpHeapEx()</code>	Creates an extended heap and can specify heap options.
<code>NNS_FndDestroyExpHeap()</code>	Destroys (deletes) an extended heap.

2.2 Allocating Memory Blocks

2.2.1 Allocating and Freeing Memory Blocks

The following functions allocate and free memory blocks.

Table 2-2 Functions for Allocating and Freeing Memory Blocks

Function	Description
<code>NNS_FndAllocFromExpHeap()</code>	Allocates memory blocks from an extended heap.
<code>NNS_FndAllocFromExpHeapEx()</code>	Allocates memory blocks from an extended heap and can specify alignment. (Described in the next section.)
<code>NNS_FndFreeToExpHeap()</code>	Frees memory blocks.

2.2.2 Minimum Allocation Unit for Memory Blocks

The Extended Heap Manager requires a 16-byte memory block management region and allocated memory blocks are aligned along a 4-byte minimum boundary. Therefore, 20 bytes of memory are required to allocate even a one-byte memory block.

2.2.3 Memory Block Allocation Procedure

The Extended Heap Manager has two modes of locating free regions from which to allocate to a memory block. You can switch between these two modes. The allocation modes are described below:

Table 2-3 Memory Block Allocation Modes

Mode	Description
FIRST Mode	Allocates a memory block from the first free region that it finds that is at least as large as the memory block size that you want to allocate.
NEAR Mode	Allocates a memory block from the free region whose size is the closest to the size of the memory block you want to allocate.

Error! Not a valid link.

Figure 2-1 Procedure for Allocating Extended Heap Memory Blocks

FIRST is the default mode. NEAR differs from FIRST in that it allocates the free block that is as close to the specified size as possible. If this mode does not find an exact fit, it searches all free regions for the closest match. Therefore, if the free regions are fragmented, memory block allocation takes longer.

The following functions set and acquire the allocation mode.

Table 2-4 Allocation Mode Setting and Acquisition

Function	Description
NNS_FndSetAllocModeForExpHeap()	Sets the allocation mode.
NNS_FndGetAllocModeForExpHeap()	Gets the current allocation mode.

The following table lists the values specified by the functions in relation to the allocation modes.

Table 2-5 Allocation Modes Used by the Functions

Mode	Value Specified with Function
FIRST mode	NNS_FND_EXPHEAP_ALLOC_MODE_FIRST
NEAR mode	NNS_FND_EXPHEAP_ALLOC_MODE_NEAR

2.2.4 Allocating Memory Blocks from the Highest Address of the Heap Region

Normally the Extended Heap Manager searches for free regions from the lowest address of the heap region to the highest address of the heap region. The Extended Heap Manager allocates memory blocks from the lowest address of the free regions that it finds. As an alternative, you can now search for free regions from the highest address of the heap region to the lowest address of the heap region. You can also allocate memory blocks from the highest address of the free regions. Using this feature, you can allocate longer-term memory blocks from the lowest address of the heap region, and temporary memory blocks from the highest address of the heap region to help minimize heap fragmentation.

For example, when loading compressed data into memory, expanding that data, then deleting the

original compressed data. If you were to use the normal method and allocate the memory block from the lowest address of the heap region, the free region will be divided into two portions.

Error! Not a valid link.

Figure 2-2 Mechanism of Memory Block Fragmentation

On the other hand, if you temporarily load the compressed data into a memory block that has been allocated from the highest address of the heap region, the free space is not split.

Error! Not a valid link.

Figure 2-3 Measures for Dealing with Extended Heap Memory Block Fragmentation

To allocate memory blocks from the highest address of the heap region, use the memory block allocation function `NNS_FndAllocFromExpHeapEx()`, passing a negative value to the alignment argument (this argument is described in the next section).

2.3 Specifying Alignment

The Extended Heap Manager specifies alignment at the time that it allocates memory blocks. You can specify the following alignment values in the `NNS_FndAllocFromExpHeapEx()` function: 4, 8, 16, and 32. To allocate memory blocks from the highest address of the heap region, specify negative alignment values (-4, -8, -16, -32). The `NNS_FndAllocFromExpHeap()` function always uses an alignment value of 4.

2.4 Changing Memory Block Size

The Extended Heap Manager can change the size of the allocated memory blocks without moving them when there is sufficient free space available. When the new memory blocks become smaller than the original size, it will use the free spaces that remain after reduction as free regions. When the new memory blocks become larger than the original size, there must be sufficient free space after the memory blocks. If there are free regions after the memory blocks, the function will merge the free regions into the memory block to increase the size of the memory block.

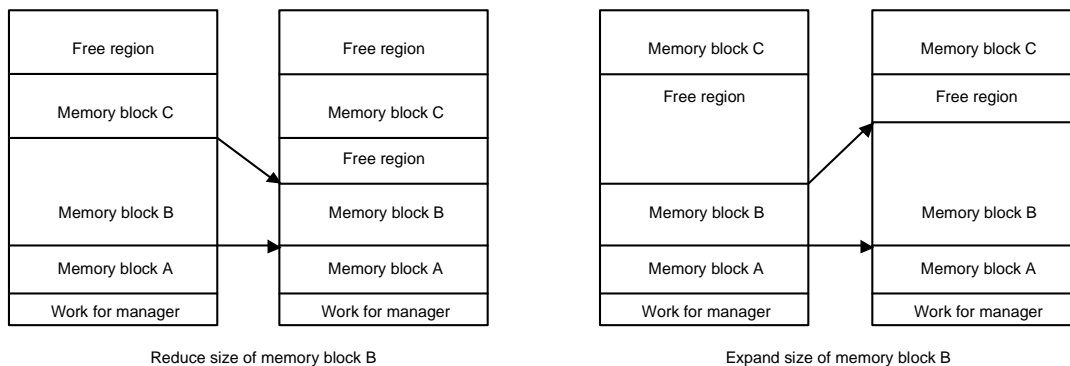


Figure 2-4 Changing Extended Heap Memory Block Size

Use this function to change the memory block size.

Table 2-6 Function for Changing Memory Block Size

Function	Description
<code>NNS_FndResizeForMBlockExpHeap()</code>	Expands or reduces memory blocks. It returns the changed memory block size.

If there is little difference between the specified memory block size and the current memory block size after a reduction, there may be times when you cannot use the freed region. In such cases, `NNS_FndResizeForMBlockExpHeap()` will not reduce the memory block size, and will return the current memory block size. If you attempt to expand memory block size when there is either no free region directly after the memory block, or if it was not possible to achieve the required expansion after defragmenting the free space behind the current memory block, `NNS_FndResizeForMBlockExpHeap()` fails and returns 0 (zero).

2.5 Acquiring Free Space

The Extended Heap Manager can obtain the total free regions. It can also obtain the size of the largest memory block that can be allocated. The functions are shown in the following table.

Table 2-7 Functions For Acquiring Free Space, Etc.

Function	Description
NNS_FndGetTotalFreeSizeForExpHeap()	Gets the total size of the free regions in the extended heap.
NNS_FndGetAllocatableSizeForExpHeap()	Gets the size of the largest memory block that can be allocated. Alignment is fixed at 4.
NNS_FndGetAllocatableSizeForExpHeapEx()	Gets the size of the largest memory block that can be allocated. You can specify alignment.

2.6 Group ID

When the Extended Heap Manager acquires memory blocks, it stores group IDs 0 - 255 in the memory block management region. You can arbitrarily change the group ID. When you change a group ID, the change takes place during the next memory block allocation. You can use the group ID for the following:

- Collectively free only memory blocks that have a specific group ID.
- Checks the memory usage for each group ID. By managing group IDs by usage or user, it becomes easier to grasp how the memory is used.

The following functions set and acquire group IDs.

Table 2-8 Functions for Setting and Acquiring Group IDs

Function	Description
NNS_FndSetGroupIDForExpHeap()	Sets extended heap group IDs.
NNS_FndGetGroupIDForExpHeap()	Acquires extended heap group IDs.

2.7 Processes for Memory Blocks

With the Extended Heap Manager, you can specify processes to be performed on allocated memory blocks. Using this functionality, it is possible to perform various processes on the heap that are not available in the Extended Heap Manager. Here are some examples:

- Call a function that collectively deletes only memory blocks that were allocated from the highest address of the heap region because they were for temporary use.
- Get the total capacity of memory blocks that have a specific group ID.

This function runs the following process.

Function	Description
NNS_FndVisitAllocatedForExpHeap()	Calls a user-specified function for each allocated memory block.

2.8 Acquiring Memory Block Information

The Extended Heap Manager can acquire information for allocated memory blocks that indicates memory block size, group ID, and whether the allocated memory blocks were allocated from the lowest address or the highest address. These functions acquire memory block information.

Table 2-9 Functions for Acquiring Memory Block Information

Function	Description
NNS_FndGetSizeForMBlockExpHeap()	Gets the memory block size.
NNS_FndGetGroupIDForMBlockExpHeap()	Gets the memory block group ID.
NNS_FndGetAllocDirForMBlockExpHeap()	Gets the direction from which the memory blocks were allocated.

2.9 Checking Heaps and Memory Blocks

With the Extended Heap Manager, extended heaps and memory blocks that are allocated from the extended heap are checked if they are destroyed. The functions that check the extended heaps and memory blocks are shown below.

Table 2-10 Functions that Check Extended Heaps and Memory Blocks

Function	Description
NNS_FndCheckExpHeap ()	Checks if the extended heap is destroyed.
NNS_FndCheckForMBlockExpHeap ()	Checks if the memory block is destroyed.

3 Frame Heap Manager

The Frame Heap Manager is an extremely simple memory manager. It can only allocate memory blocks in a specified size while simultaneously freeing all allocated memory blocks. Since it holds no memory block management information, it is memory-efficient. This section gives an overview of the Frame Heap Manager.

3.1 Creating Heaps

To use the Frame Heap Manager you must create a frame heap. These functions create and destroy frame heaps.

Table 3-1 Functions for Creating and Destroying Heaps

Function	Description
NNS_FndCreateFrmHeap()	Creates a frame heap.
NNS_FndCreateFrmHeapEx()	Creates a frame heap. You can specify options for the heap.
NNS_FndDestroyFrmHeap()	Destroys a frame heap.

3.2 Allocating Memory Blocks

3.2.1 Allocating and Freeing Memory Blocks

The Frame Heap Manager allocates memory blocks by packing them with no open space from the lowest address and the highest address of the heap region. Because it allocates memory blocks this way, it does not fragment the heap. Also, because there is no management region in the memory blocks that the Frame Heap Manager allocates, memory usage is efficient and less processing is required to allocate the memory blocks.

Error! Not a valid link.

Figure 3-1 Frame Heap Memory Allocation

The following functions allocate memory blocks.

Table 3-2 Functions for Allocating Memory Blocks

Function	Description
NNS_FndAllocFromFrmHeap()	Allocates memory blocks from a frame heap.
NNS_FndAllocFromFrmHeapEx()	Allocates memory blocks from a frame heap. It is possible to specify alignment (described in the next section).

Use negative alignment numbers in the NNS_FndAllocFromFrmHeapEx() function to allocate memory from the highest address of the heap region.

3.2.2 The Minimum Unit of Memory Block Allocation

Although the Frame Heap Manager does not have a management region in the memory blocks, memory blocks must be 4-byte aligned. Therefore, even allocating a 1-byte memory block uses 4 bytes of memory.

3.3 Specifying Alignment

The Frame Heap Manager can specify alignment when it allocates memory blocks. You can specify the following alignment values in the `NNS_FndAllocFromFrmHeapEx()` function: 4, 8, 16, and 32. To allocate memory blocks from the highest address of the heap region, specify negative alignment values (-4, -8, -16, and -32). The `NNS_FndAllocFromFrmHeap()` function does not specify alignment, it is always a value of 4.

3.4 Freeing Memory Blocks

Because the Frame Heap Manager does not manage individual allocated memory blocks, it cannot free allocated blocks individually. The Frame Heap Manager uses one of the three following methods to free memory blocks.

Table 3-3 Methods Of Freeing Memory Blocks

Freeing method	Description
Free from the lowest address	Collectively frees memory blocks that were allocated from the lowest address of the heap region.
Free from the highest address	Collectively frees memory blocks that were allocated from the highest address of the heap region.
Free all	Collectively frees all of the memory blocks that were allocated from the heap.

The following function frees memory blocks.

Table 3-4 Function for Freeing Memory Blocks

Function	Description
<code>NNS_FndFreeToFrmHeap()</code>	Collectively frees memory blocks using the method specified.

The following methods are specified in the `NNS_FndFreeToFrmHeap()` function.

Table 3-5 Values Specified in the Function for Freeing Memory Blocks

Freeing method	Value to Specify in the Function
Free from the lowest address	<code>NNS_FND_FRMHEAP_FREE_HEAD</code>
Free from the highest address	<code>NNS_FND_FRMHEAP_FREE_TAIL</code>
Free all	<code>NNS_FND_FRMHEAP_FREE_ALL</code> (Same as simultaneously specifying <code>NNS_FND_FRMHEAP_FREE_HEAD</code> and <code>NNS_FND_FRMHEAP_FREE_TAIL</code>)

The Frame Heap Manager also offers you the option of saving the memory block allocation status, collectively freeing subsequently allocated memory blocks, and returning to the status immediately prior to saving. These options are described in the next section.

3.5 Saving and Restoring a Memory Block Allocation Status

The Frame Heap Manager offers you the option of saving the memory block allocation status of the heap region and restoring that status later.

20 bytes of memory are required to save each memory block allocation status. You can store memory block allocation status as many times as you want to the limit of heap capacity. When saving the memory block allocation status, a 4-byte tag can be attached. When you restore a memory block allocation status, you can return to the previous status or to a status that is specified by the flag.

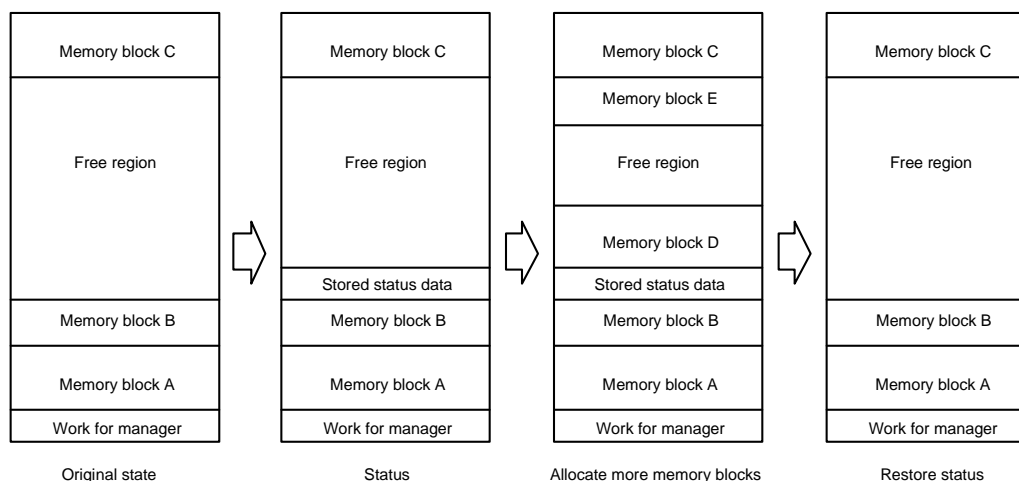


Figure 3-2 Mechanism for Saving and Restoring Frame Heap Memory Block Allocation Status

The following functions store memory block allocation status and return to it.

**Table 3-6 Functions for Storing a
Memory Block Allocation Status and Returning to It**

Function	Description
<code>NNS_FndRecordStateForFrmHeap()</code>	Saves a memory block allocation status.
<code>NNS_FndFreeByStateToFrmHeap()</code>	Restores a memory block allocation status.

3.6 Adjusting Heap Region Size

The Frame Heap Manager can reduce the heap region size to match the heap region content. However, use this functionality only if no memory blocks have been allocated from the highest address of the heap region.

Use this functionality when you want to pack an indefinite amount of data into memory without leaving spaces in the heap. First create a heap that has a sufficient size, allocate memory blocks from the lowest address of the heap region, and store the data. After storing all of the required data, reduce the size of the heap region to match the contents of the heap.

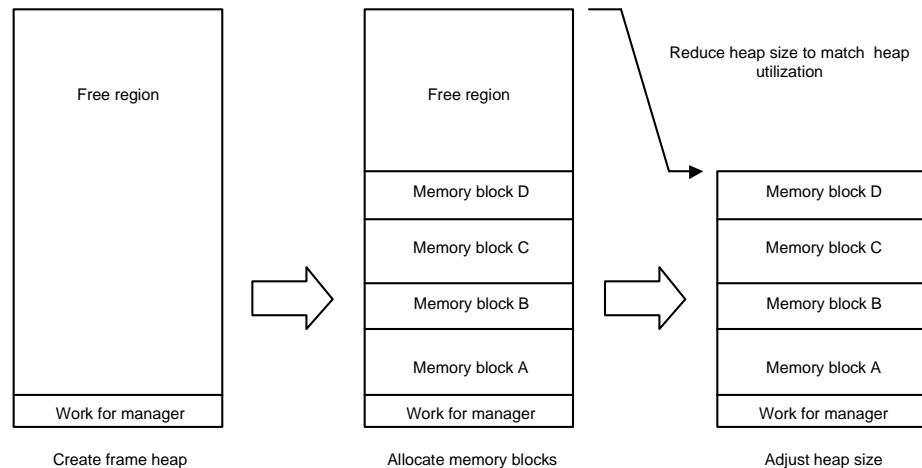


Figure 3-3 Adjusting Frame Heap Size

The following function reduces the heap region size.

Table 3-7 Function for Reducing Heap Region Size

Function	Description
<code>FndAdjustFrmHeap()</code>	Reduces the size of the heap region by freeing space at the highest address of the heap region from the allocated block.

3.7 Changing the Size of Memory Blocks

You can change the size of the memory block with the frame heap manager if it is the last memory block that is allocated from the lowest address of the empty region in the heap. When the memory block is reduced, the remaining area after reduction becomes part of the empty area. When the memory block is enlarged, it reduces the free region of the higher address and expands the memory block.

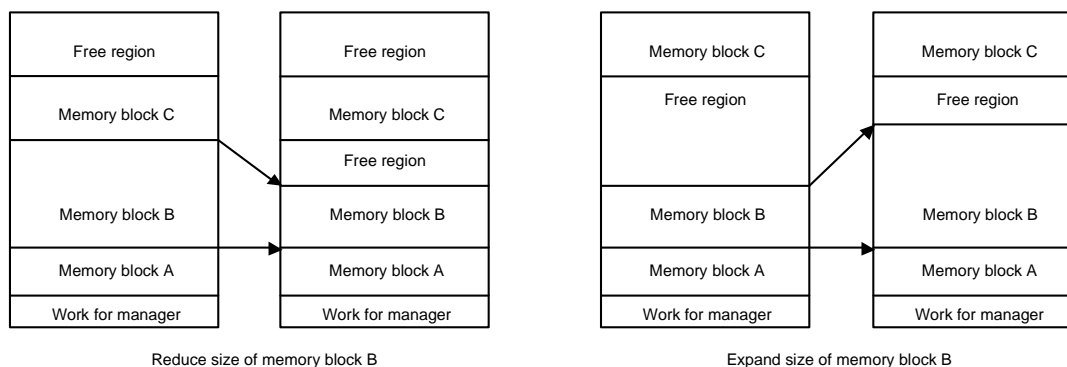


Figure 3-4 Adjusting the size of the frame heap memory block

The following function changes the size of memory block.

Table 3-8 Function for Changing the Size of Memory Block

Function	Description
<code>NNS_FndResizeForMBlockFrmHeap()</code>	Expands or reduces the memory block. The changed memory block size is returned as the return value.

When expanding a memory block, if there is not enough free space to expand to the requested size, the `NNS_FndResizeForMBlockExpHeap()` function fails and returns 0.

3.8 Acquiring the Size That Can Be Allocated

The Frame Heap Manager can find the size of the largest memory block that can be allocated. The following functions do this.

Table 3-9 Functions for Obtaining the Size That Can Be Allocated

Function	Description
<code>NNS_FndGetAllocatableSizeForFrmHeap()</code>	Gets the size of the largest memory block that can be allocated. Alignment is fixed at 4.
<code>NNS_FndGetAllocatableSizeForFrmHeapEx()</code>	Gets the size of the largest memory block that can be allocated. You can specify alignment.

4 Unit Heaps

The Unit Heap Manager is an extremely simple memory manager. It allocates only memory blocks in the size that is specified when the unit heap is created. In other words, this memory manager is for allocating and freeing memory blocks that have a fixed size. The unit heap does not have a management region in memory blocks, making it more memory-efficient. This section provides an overview of the Unit Heap Memory Manager.

4.1 Creating Heaps

To use the Unit Heap Manager you must create a unit heap. The following functions create and destroy (delete) unit heaps.

Table 4-1 Functions for Creating and Destroying Heaps

Function	Description
NNS_FndCreateUnitHeap()	Creates a unit heap.
NNS_FndCreateUnitHeapEx()	Creates a unit heap. You can specify alignment and options for the heap.
NNS_FndDestroyUnitHeap()	Destroys (deletes) a unit heap.

4.2 Memory Block Allocation

4.2.1 Allocating and Freeing Memory Blocks

The Unit Heap Manager manages the regions in a heap in chunks that are of a pre-specified size. Memory block allocation means allocating these chunks.

Free chunks are linked as a singly-linked list (free chunk list). The pointer to the next free chunk is placed at the beginning of the chunk. There is no pointer for chunks that are in use. (There is also no management region.) When allocating memory blocks, the manager returns the memory block that is linked to the beginning of this free chunk list. When freeing a memory block that is in use, it links the memory block to the beginning of the free chunk list.

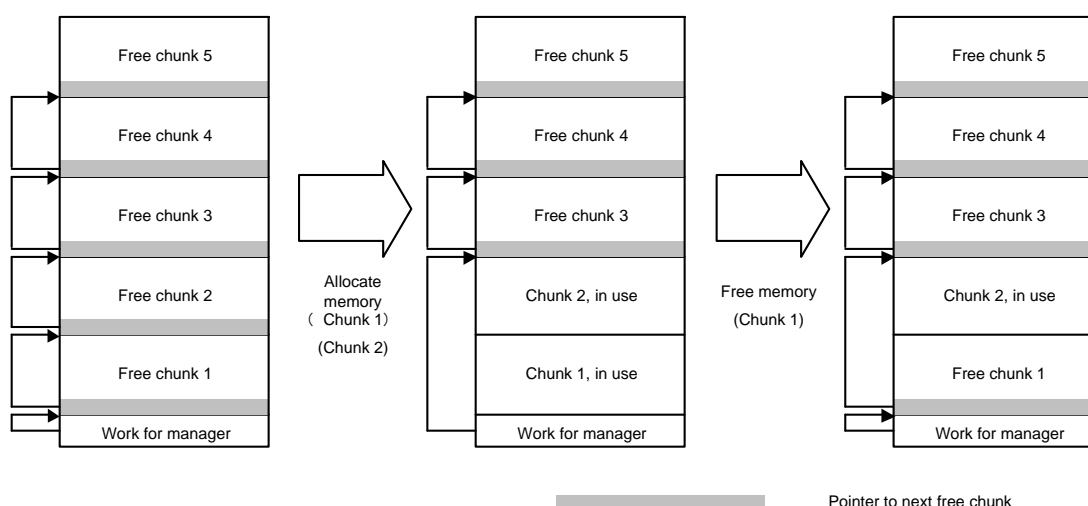


Figure 4-1 Unit Heap Memory Allocation

The following functions allocate and free memory blocks.

Table 4-2 Function for Allocating Memory Blocks

Function	Description
<code>NNS_FndAllocFromUnitHeap()</code>	Allocates memory blocks from a unit heap.
<code>NNS_FndFreeToUnitHeap()</code>	Frees memory blocks.

4.2.2 Minimum Unit for Memory Block Allocation

Although the Unit Heap Manager does not have a management region in the memory blocks, allocated memory blocks must be 4-byte aligned. Therefore, even allocating a 1-byte memory block uses 4 bytes of memory.

4.3 Specifying Alignment

The Unit Heap Manager can specify alignment when it creates a heap. It does not do this for each allocated memory block. All allocated memory blocks will be aligned the same. You can specify the following alignment values in the `NNS_FndCreateUnitHeapEx()` function: 4, 8, 16, and 32. The `NNS_FndCreateUnitHeap()` function does not specify alignment, its alignment is always 4.

4.4 Acquiring the Number of Memory Blocks That Can Be Allocated

The Unit Heap Manager can get the number of memory blocks that can be allocated. In other words, it can get the number of free chunks. The following function gets the number of memory blocks that can be allocated.

Table 4-3 Function for Acquiring the Number of Memory Blocks That Can Be Allocated

Function	Description
<code>NNS_FndCountFreeBlockForUnitHeap()</code>	Returns the number of memory blocks that can be allocated.

5 Functionality Common to Each Heap

This section describes the functionality that is common to the extended heap, the frame heap, and the unit heap.

5.1 Heap Options

Three of the functions that create heaps specify heap options (`NNS_FndCreateExpHeapEx()`, `NNS_FndCreateFrmHeapEx()`, and `NNS_FndCreateUnitHeapEx()`). You can specify the following options.

Table 5-1 Options That Can Be Specified When Creating a Heap

Flag	Description
<code>NNS_FND_HEAP_OPT_0_CLEAR</code>	When memory is allocated from a heap, this fills the allocated memory blocks with 0s.
<code>NNS_FND_HEAP_OPT_DEBUG_FILL</code>	When a heap is created and memory blocks are allocated and freed, this fills the memory regions respectively with different 32-bit values.

The flag `NNS_FND_HEAP_OPT_DEBUG_FILL` was made for debugging. Use it to find memory access bugs by tracing the pointer that points to the memory initialization failure or invalid memory regions. This will not function in the final ROM version (`FINALROM`) library.

The following values are filled into the memory regions by default. See the next section for instructions on how to change them.

- When creating a heap `0xC3C3C3C3`
- When allocating memory `0xF3F3F3F3`
- When freeing memory `0xD3D3D3D3`

5.2 Changing the Values to Fill When Debugging

When the heap is created, and when memory blocks are allocated and freed, you will be able to fill each memory region with different 32-bit values if you specify the `NNS_FND_HEAP_OPT_DEBUG_FILL`. You can set and get the fill values using the following functions.

Table 5-2 Functions for Setting and Acquiring Values to Fill when Debugging

Function	Description
<code>NNS_FndSetFillValForHeap()</code>	Sets fill values.
<code>NNS_FndGetFillValForHeap()</code>	Acquires fill values.

You can set different values for creating heaps, allocating memory blocks, and freeing memory blocks. Specify which heap operation the value is for, when you set or acquire values. The following table shows the types of heap operations that are specified in the function.

Table 5-3 Type of Heap Operation for Filling the Value

Value to Specify in the Function	Heap Operation
NNS_FND_HEAP_FILL_NOUSE	When creating a heap.
NNS_FND_HEAP_FILL_ALLOC	When allocating memory blocks.
NNS_FND_HEAP_FILL_FREE	When freeing memory blocks.

5.3 Displaying Heap Contents

This feature is for debugging. It displays internal heap information.

Table 5-4 Function for Displaying Internal Heap Information

Function	Description
NNS_FndDumpHeap	Displays internal heap information.

5.4 Acquiring Heap Regions

This feature obtains the start and end addresses of the memory region that a heap is using.

Table 5-5 Functions For Acquiring Heap Regions

Function	Description
NNS_FndGetHeapStartAddress()	Gets the start address of the memory region that the heap is using.
NNS_FndGetHeapEndAddress()	Gets the end address (+1) of the memory region that the heap is using.

6 Multi-Heap Management

This section covers instances in which the game software creates and uses multiple heaps.

6.1 Multi-Heaps

There are various types of data that are used during a game, such as data for graphics, music, and the system. Using multiple heaps such as a game heap, a sound heap, and a system heap will make it easier to manage this data. Such use of multiple heaps is referred to as multi-heaps.

6.2 Freeing Multi-Heap Memory

A programmer should know from which heap to allocate memory blocks. Therefore, the programmer can specify a heap and allocate memory blocks from it.

What about freeing memory blocks? If these are memory blocks that you have allocated, you will know to which heap you should return the memory blocks. Even in the case where you free memory blocks received from another programmer, you will probably be able to determine the correct heap, as long as the use of the multiple heaps is clearly defined. However, what do you do when multiple heaps are candidates for the return of memory blocks?

6.3 Managing Heaps Using a Tree Structure

When you free memory blocks and you do not know from where they were allocated, it would be convenient if there were a mechanism that searched for the heap from which the memory block had been allocated. You can do this by managing the heaps with a tree structure. This allows you to use memory blocks allocated from a heap as heap memory (hierarchical heap structure).

If you manage heaps with trees, you can recursively check the memory region that the heap occupies, thus checking from which heap the memory block was allocated.

The NITRO-System memory manager internally creates a hierarchical structure for each heap that it creates. There is also a function that searches for the heap from which a memory block has been allocated.

Table 6-1 Function for Searching for the Heap That Allocated a Memory Block

Function	Description
NNS_FndFindContainHeap()	Searches for the heap from which the specified memory block was allocated. Returns a handle to the found heap.

© 2004-2007 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed or loaned in whole or in part without the prior approval of Nintendo.