1. Libraries

1. libc

1.1. Include files

1.1.1. stdtypes.h

1.1.1.1. Defines

```
#define __NEOGEO__
```
The platform we are compiling for.

```
#define NULL      0
```
The famous NULL value.

```
#define TRUE            1
#define FALSE     0
```
For boolean operations.

```
#define min(a,b) ((a) < (b) ? (a) : (b))
#define max(a,b) ((a) > (b) ? (a) : (b))
```
Two macros that will return the minimum or the maximum of two values.

```
#define __PACKED__      __attribute__ ((packed))
```
Change alignment of structure or array to 1.

```
#define __ALIGN16__     __attribute__ ((aligned (16)))
```
Change alignment of structure or array to 16.

```
#define __ALIGN32__     __attribute__ ((aligned (32)))
```
Change alignment of structure or array to 32.

```
#define __NORETURN__    __attribute__ ((noreturn))
```
To declare a function that will never return

```
#define __CONSTRUCTOR__ __attribute__ ((constructor))
```
To declare a special function that will be called before main()

```
#define __DESTRUCTOR__  __attribute__ ((destructor))
```
To declare a special function that will be called after main()

1.1.1.2. Typedefs

```
typedef unsigned char         BYTE, *PBYTE;
```
Unsigned 8 bits value.

```
typedef unsigned short        WORD, *PWORD;
```
Unsigned 16 bits value.

```
typedef unsigned int          DWORD, *PDWORD;
```
Unsigned 32 bits value.

```
typedef unsigned long long    UQUAD, *PUQUAD;
```
Unsigned 64 bits value.

```
typedef long long             INT64;
```
Signed 64 bits value.

```
typedef unsigned int          BOOL, *PBOOL;
```
For boolean operations

```
typedef unsigned int          size_t;
```
Type for size specification (=int)

1.1.2. ctype.h

Nothing.

1.1.3. limits.h

Include file is self explanatory. Read it for more information.

1.1.4. stdio.h

Nothing.

1.1.5. stdlib.h

Nothing

1.1.6. string.h, strings.h

Nothing

1.2. Functions

1.2.1. isascii

Syntax
```
#include <ctype.h>
int isascii(int c);
```

Description
Tells if c is an ASCII character (0x00 to 0x7f).

Return Value
Nonzero if c is ASCII, else zero.

1.2.2. isdigit

Syntax
```
#include <ctype.h>
int isdigit(int c);
```

Description
Tells if c is a digit.

Return Value
Nonzero if c is a digit, else zero.

1.2.3. islower

Syntax
```
#include <ctype.h>
int islower(int c);
```

Description
Tells if c is lower case or not.

Return Value
Nonzero if c is lower case, else zero.

1.2.4. isupper

Syntax
```
#include <ctype.h>
int isupper(int c);
```

Description
Tells if c is an upper case character or not.

Return Value
Nonzero if c is upper case, else zero.

1.2.5. isxdigit

Syntax
#include <ctype.h>
int isxdigit(int c);

Description
Tells if c is a valid hexidecimal digit or not. This includes `[0-9a-fA-F]'.

Return Value
Nonzero if c is a hex digit, else zero.

1.2.6. memcpy

Syntax
```
#include <string.h>
void *memcpy(void *dest, const void *src, int num);
```

Description
This function copies num bytes from source to dest.

Return Value
dest

Example
```
memcpy(buffer, temp_buffer, BUF_MAX);
```

1.2.7. memmove

Syntax
```
#include <string.h>
void *memmove(void *dest, const void *source, int num);
```

Description
This function copies num bytes from source to dest. The copy is done in such a way that if the two regions overlap, the source is always read before that byte is changed by writing to the destination.

Return Value
dest

Example
```
memmove(buf+1, buf, 99);
memmove(buf, buf+1, 99);
```

1.2.8. memset

Syntax
```
#include <string.h>
void *memset(void *buffer, int ch, size_t num);
```

Description
This function stores num copies of ch, starting at buffer. This is often used to initialize objects to a known value.

Return Value
buffer

Example
```
struct tm t;
memset(&t, 0, sizeof(t));
```

1.2.9. rand

Syntax
```
#include <stdlib.h>
int rand(void);
```

Description
Returns a pseudo-random number from zero to `RAND_MAX'.

Return Value
The number.

Example
```
/* random pause */
for (i=rand(); i; i--);
```

1.2.10. sprintf

Syntax
```
#include <stdio.h>
int sprintf(char *buffer, const char *format, ...);
```

Description
Sends formatted output from the arguments (...) to the buffer.

The format string contains regular characters to print, as well as conversion specifiers, which begin with a percent symbol. Each conversion specifier contains the following fields:

- An optional flag, which may alter the conversion:

Flag	Description
`-'	Left-justify the field.
`+'	Force a `+' sign on positive numbers.
`space'	To leave a blank space where a plus or minus sign would have been.

`0'	To pad numbers with leading zeros.

- A field width specifier, which specifies the minimum width of the field. This may also be an asterisk (`*'), which means that the actual width will be obtained from the next argument. If the argument is negative, it supplies a `-' flag and a positive width.

- An optional decimal point and a precision. This may also be an asterisk, but a negative argument for it indicates a precision of zero. The precision specifies the minimum number of digits to print for an integer, the number of fraction digits for a floating point number, or the maximum number of characters for a string.

- An optional conversion qualifier, which may be `h' to specify `short', `l' to specify long ints, or `L' to specify long doubles. Long long type can be specified by `L' or `ll'. Qualifiers are ignored on NeoGeo.

- The conversion type specifier:

Specifier	Description
`c'	Single character
`d'	Signed integer
`i'	A signed integer.
`n'	The next argument is a pointer to an integer.
`p'	A pointer. This is printed with an `x' specifier.
`s'	A zero-terminated string.
`u'	An unsigned integer.
`x', `X'	An unsigned integer, printed in base 16 instead of base 10. The case of the letters used matches the specifier case.
`%'	A single percent symbol is printed.

Return Value
The number of characters written.

1.2.11. srand

Syntax
```
#include <stdlib.h>
int srand (int seed);
```

Description
This function initialized the random number generator (see rand). Passing the same seed results in `rand' returning predictable sequences of numbers.

Return Value
Zero.

Example
```
srandom(45);
```

1.2.12. strcat

Syntax
```
#include <string.h>
char *strcat(char *s1, const char *s2);
```

Description
This function concatenates s2 to the end of s1.

Return Value
s1

Example
```
char buf[100] = "hello";
strcat(buf, " there");
```

1.2.13. strchr

Syntax
```
#include <string.h>
char *strchr(const char *s, int c);
```

Description
This function returns a pointer to the first occurrence of c in s. Note that if c is zero, this will return a pointer to the end of the string.

Return Value
A pointer to the character, or `NULL' if it wasn't found.

Example
```
char *slash = strchr(filename, '/');
```

1.2.14. strcmp

Syntax
```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

Description
This function compares s1 and s2.

Return Value
Zero if the strings are equal, a positive number if s1 comes after s2 in the ASCII collating sequence, else a negative number.

Example
```
if (strcmp(arg, "-i") == 0)
      do_include();
```

1.2.15. strcpy

Syntax
```
#include <string.h>
char *strcpy(char *s1, const char *s2);
```

Description
This function copies s2 into s1.

Return Value
s1

Example
```
char buf[100];
strcpy(buf, arg);
```

1.2.16. strlen

Syntax
```
#include <string.h>
size_t strlen(const char *string);
```

Description
This function returns the number of characters in string.

Return Value
The length of the string.

Example
```
if (strlen(fname) > PATH_MAX)
      invalid_file(fname);
```

1.2.17. strncat

Syntax
```
#include <string.h>
char *strncat(char *s1, const char *s2, size_t max);
```

Description
This function concatenates up to max characters of s2 to the end of s1.

Return Value
s1

Example
```
strncat(fname, extension, 4);
```

1.2.18. strncmp

Syntax
```
#include <string.h>
int strncmp(const char *s1, const char *s2, size_t max);
```

Description
This function compares upto max characters of s1 and s2.

Return Value
Zero if the strings are equal, a positive number if s1 comes after s2 in the ASCII collating sequence, else a negative number.

Example
```
if (strncmp(arg, "-i", 2) == 0)
      do_include();
```

1.2.19. strncpy

Syntax
```
#include <string.h>
char *strncpy(char *s1, const char *s2, size_t max);
```

Description
This function copies up to max characters of s2 into s1.

Return Value
s1

Example
```
char buf[100];
strcpy(buf, arg, 99);
```

1.2.20. strnlen

Syntax
```
#include <string.h>
size_t strlen(const char *string, size_t max);
```

Description
This function returns the number of characters in string, limited to max chars.

Return Value
The length of the string.

1.2.21. strpbrk

Syntax
```
#include <string.h>
char *strpbrk(const char *s1, const char *set);
```

Description
This function finds the first character in s1 that matches any character in set.

Return Value
A pointer to the first match, or `NULL' if none are found.

Example
```
if (strpbrk(command, "<>|"))
      do_redirection();
```

1.2.22. strrchr

Syntax
```
#include <string.h>
char *strrchr(const char *s1, int c);
```

Description
This function finds the last occurrence of `c' in `s1'.

Return Value
A pointer to the last match, or `NULL' if the character isn't in the string.

Example
```
char *last_slash = strrchr(filename, '/');
```

1.2.23. strspn

Syntax
```
#include <string.h>
size_t strspn(const char *s1, const char *set);
```

Description
This function finds the first character in s1 that does not match any character in set. Note that the zero byte at the end of s1 counts, so you'll at least get a pointer to the end of the string if nothing else.

Return Value
The index of the found character.

Example
```
int i = strcspn(entry, " \t\b");
if (entry[i])
      do_something();
```

1.2.24. strstr

Syntax
```
#include <string.h>
char *strstr(const char *s1, const char *s2);
```

Description
This function finds the first occurrence of s2 in s1.

Return Value
A pointer within s1, or `NULL' if s2 wasn't found.

Example
```
if (strstr(command, ".exe"))
      do_exe();
```

1.2.25. strtok

Syntax
```
#include <string.h>
char *strtok(char *s1, const char *s2);
```

Description
This function retrieves tokens from s1 which are delimited by characters from s2.
To initiate the search, pass the string to be searched as s1. For the remaining tokens, pass `NULL' instead.

Return Value
A pointer to the token, or `NULL' if no more are found.

Example
```
main()
{
      char *buf = "Hello there, stranger";
      char *tok;
      for (tok = strtok(buf, " ,");tok;tok=strtok(0, " ,"))
            printf("tok = `%s'\n", tok);
}

tok = `Hello'
tok = `there'
tok = `stranger'
```

1.2.26. toascii

Syntax
```
#include <ctype.h>
int toascii(int c);
```

Description
This function strips the high bit of c, forcing it to be an ASCII character.

Return Value
The ASCII character.

Example
```
for (i=0; buf[i]; i++)
      buf[i] = toascii(buf[i]);
```

1.2.27. tolower

Syntax
```
#include <ctype.h>
int tolower(int c);
```

Description
This function returns c, converting it to lower case if it is upper case. See toupper.

Return Value
The lower case letter.

Example
```
for (i=0; buf[i]; i++)
      buf[i] = tolower(buf[i]);
```

1.2.28. toupper

Syntax
```
#include <ctype.h>
int toupper(int c);
```

Description
This function returns c, converting it to upper case if it is lower case..

Return Value
The upper case letter.

Example
```
for (i=0; buf[i]; i++)
      buf[i] = toupper(buf[i]);
```

1.2.29. vsprintf

Syntax
```
#include <stdio.h>
#include <stdarg.h>
int vsprintf(char *buffer, const char *format, va_list arguments);
```

Description
Sends formatted output from the arguments to the buffer. See sprintf.

Return Value
The number of characters written.

1.2.30. malloc (Neo Geo CD only)

Syntax
```
#include <stdlib.h>
void *malloc(size_t size);
```

Description
This function allocates a chunk of memory from the heap large enough to hold any object that is size bytes in length. This memory must be returned to the heap with `free' (see free).

Return Value
A pointer to the allocated memory, or `NULL' if there isn't enough free memory to satisfy the request.

Example
```
char *c = (char *)malloc(100);
```

1.2.31. free (Neo Geo CD only)

Syntax
```
#include <stdlib.h>
void free(void *ptr);
```

Description
Returns the allocated memory to the heap (see malloc). If the ptr is `NULL', it does nothing.

Return Value
None.

Example
```
char *q = (char *)malloc(20);
free(q);
```

1.2.32. calloc (Neo Geo CD only)

Syntax
```
#include <stdlib.h>
void *calloc(size_t num_elements, size_t size);
```

Description
This function allocates enough memory for num_elements objects of size size. The memory returned is initialized to all zeros. The pointer returned should later be passed to free (see free) so that the memory can be returned to the heap.

Return Value
A pointer to the memory, or `NULL' if no more memory is available.

Example
```
Complex *x = calloc(12, sizeof(Complex));
free(x);
```

1.2.33. realloc (Neo Geo CD only)

Syntax
```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

Description
This function changes the size of the region pointed to by ptr. If it can, it will reuse the same memory space, but it may have to allocate a new memory space to satisfy the request. In either case, it will return the pointer that you should use to refer to the (possibly new) memory area. The pointer passed may be `NULL', in which case this function acts just like `malloc' (see malloc).

Return Value
A pointer to the memory you should now refer to.

Example
```
if (now+new > max)
{
  max = now+new;
  p = realloc(p, max);
}
```

1.2.34. memalign (Neo Geo CD only)

Syntax
```
#include <stdlib.h>
void* memalign(size_t align, size_t bytes);
```

Description
This function allocates a chunk of memory from the heap large enough to hold any object that is size bytes in length. The returned memory block will have the specified alignment (2, 4, 8, 16, …).
This memory must be returned to the heap with `free' (see free).

Return Value
A pointer to the aligned allocated memory, or `NULL' if there isn't enough free memory to satisfy the request.

Example
```
Complex *x = memalign(32, sizeof(Complex));
```

2. libinput

2.1. Predefined types and structures

None.

2.2. Functions

2.2.1. poll_joystick

Syntax
```
#include <input.h>
```

```
DWORD poll_joystick(DWORD port, DWORD flags);
```

Description
Returns current joystick position for the specified port using the specified method.

Valid values for port parameter are :

Value	Description
PORT1	Read the first port
PORT2	Read the second port

Valid values for flags parameter are :

Value	Description
READ_DIRECT	Directly read hardware
READ_BIOS	Read thru BIOS
READ_BIOS_CHANGE	Read thru BIOS. Report position only if it changes
READ_BIOS_REPEAT	Read thru BIOS. Report position at regular intervals

Return Value
Current joystick position. You can use the following constants to decode it :

Value
JOY_UP
JOY_DOWN
JOY_LEFT
JOY_RIGHT
JOY_A
JOY_B
JOY_C
JOY_D
JOY_START
JOY_SELECT

Example

```
joy1_pos = poll_joystick(PORT1, READ_BIOS);

if (joy1_pos & JOY_UP)
      …
```

Note
Auto-Repeat feature is not available for start & select buttons

3. libmath

Note
The 68000 is awfully slow when it comes to math functions. Use precalc tables instead wherever you can.

3.1. Predefined types and structures

```
typedef int FIXED;
```
This defines the fixed point number. Please note that fixed point numbers have special properties and can't be divided or multiplied using normal operators.

To make a fixed point number constant to use in your program, simply multiply a floating point number by 65536 and round the result. Let's say I want to use PI in my program :
```
3.1415926 * 65536 = 205887.4  => 205887 ($3243F)
```

So I can write :
```
FIXED pi;
pi = 0x3243F;
```
(In that case 'pi' is exactly 3.141586)

3.2. Functions

3.2.1. fadd

Syntax
```
#include <math.h>
FIXED fadd(FIXED x, FIXED y);
```

Description
Adds two fixed point values.
This function does clamping on the result value.
If you don't need the clamping, you can directly add fixed point values using normal operators.

Return Value
Fixed point value of the sum.

3.2.2. fcos

Syntax
```
#include <math.h>
FIXED fcos(int x);
```

Description
Returns the cosinus of the specified angle.
Please note that to be « more computer friendly » in this library angles are 0..512 and not 0..360

Return Value
Fixed point value of the cosinus.

3.2.3. fixtof

Syntax
```
#include <math.h>
double fixtof(FIXED x);
```

Description
Converts a fixed point value into a floating point value.

Return Value
Floating point value.

3.2.4. fixtoi

Syntax
```
#include <math.h>
int fixtoi(FIXED x);
```

Description
Converts a fixed point value into an integer.

Return Value
(Rounded) integer value.

3.2.5. fmul

Syntax
```
#include <math.h>
FIXED fmul(FIXED a, FIXED b);
```

Description
Compute the product of two fixed point values.

Note
You can’t use normal multiplication operator on fixed point values.

Return Value
Fixed point value of the product.

3.2.6. fmuli

Syntax
```
#include <math.h>
FIXED fmuli(FIXED a, WORD b);
```

Description
Compute the product of a fixed point value and an integer.

Note
You can’t use normal multiplication operator on fixed point values.

Return Value
Fixed point value of the product.

3.2.7. fsin

Syntax
```
#include <math.h>
FIXED fsin(int x);
```

Description
Returns the sinus of the specified angle.
Please note that to be « more computer friendly » in this library angles are 0..512 and not 0..360

Return Value
Fixed point value of the sinus.

3.2.8. fsub

Syntax
```
#include <math.h>
FIXED fsub(FIXED x, FIXED y);
```

Description
Substracts two fixed point values.
This function does clamping on the result value.
If you don’t need the clamping, you can directly substract fixed point values using normal operators.

Return Value
Fixed point value of the difference.

3.2.9. ftan

Syntax
```
#include <math.h>
FIXED ftan(int x);
```

Description
Returns the tangent of the specified angle.
Please note that to be « more computer friendly » in this library angles are 0..512 and not 0..360

Return Value
Fixed point value of the tangent.

3.2.10. ftofix

Syntax
```
#include <math.h>
FIXED ftofix(double x);
```

Description
Converts a floating point value into a fixed point value.

Return Value
Fixed point value.

3.2.11. ifmuli

Syntax
```
#include <math.h>
DWORD ifmuli(FIXED a, WORD b);
```

Description
Compute the product of a fixed point value and an integer, an return the result in the form of an integer.

Note
You can’t use normal multiplication operator on fixed point values.

Return Value
(Rounded) integer value of the product.

3.2.12. itofix

Syntax
```
#include <math.h>
FIXED itofix(int x);
```

Description
Converts an integer into a fixed point value.

Return Value
Fixed point value.

4. libprocess

4.1. Defines

```
// Maximum number of tasks
#ifndef MAX_TASKS
#define MAX_TASKS       16
#endif

// Stack size for all tasks
#ifndef STACK_SIZE
#define STACK_SIZE      1024
#endif
```

You'll notice that those two defines can be 'overloaded'. If you need special values for these parameters (more stack room or more tasks) you can define these two values in your main source file before including « task.h ». However, the process library is compiled for these values, so if you change them, you'll have to directly include task.c and mutex.c in your project instead of linking libprocess.

```
// Task states
#define TASK_TERMINATED 0
#define TASK_RUNNING    1
#define TASK_SUSPENDED  2

// Task enumeration criterias

// Enumerate tasks that match tag1
#define TASKENUM_TAG1   1
// Enumerate tasks that match tag2
#define TASKENUM_TAG2   2
// Enumerate tasks that match both tags
#define TASKENUM_BOTH   3
// Enumerate tasks that match the state
#define TASKENUM_STATE  4
// Enumerate that match all attributes
#define TASKENUM_ALL    7
// Enumerate all tasks
#define TASKENUM_NONE   0

// Macro to make 'text' identifiers
#define MAKE_ID(a,b,c,d)     (((a)<<24) | ((b)<<16) | ((c)<<8) | (d))

// Mailbox error codes
#define MAILBOX_OK            0
#define MAILBOX_NO_MESSAGE    1
#define MAILBOX_CANT_POST     2
```

4.2. Defined types and structures

```
// CPU context structure
typedef struct __PACKED__ {
      DWORD a7;
      DWORD pc;
      WORD  sr;
```

```
	DWORD d0;
	DWORD d1;
	DWORD d2;
	DWORD d3;
	DWORD d4;
	DWORD d5;
	DWORD d6;
	DWORD d7;
	DWORD a0;
	DWORD a1;
	DWORD a2;
	DWORD a3;
	DWORD a4;
	DWORD a5;
	DWORD a6;
} CPU_CONTEXT, *PCPU_CONTEXT;
```

CPU context containing all registers. This should NOT be modified.

```
// Mailbox structure
typedef struct {
	void  *from;
	DWORD data1;
	DWORD data2;
} MAILBOX, *PMAILBOX;
```

Used to send messages from task to task

```
// Task structure
typedef struct _TASK {
	// CPU context of the task
	CPU_CONTEXT	context;

	// State of the task (Terminated, running, suspended)
	DWORD		state;

	// Priority (can only be defined when the task is created)
	DWORD		prio;

	// User tag for the task (1). Can be used with task_enum.
	DWORD		tag1;

	// User tag for the task (2). Can be used with task_enum.
	DWORD		tag2;

	// Task "mailbox" for inter-task communication
	MAILBOX		mailbox;

	// Pointer to user shared data structure
	void		*data;

	// Stack reserved for the task
	BYTE		stack[STACK_SIZE];

	// Pointer to next task
	struct _TASK	*next;
} TASK, *PTASK;
```

The TASK structure itself. Most of this structure can be changed « manually » if you wish so except context, prio and next.

4.3. Functions

4.3.1. _release_timeslice

Syntax
```
#include <task.h>
void _release_timeslice(void);
```

Description
You must call this function regularly in your task routine to let other tasks execute.

Return Value
None.

4.3.2. mutex_clean

Syntax
```
#include <task.h>
void mutex_clean(PTASK task);
```

Description
Free all mutexes owned by the specified task. Called by task_kill.

Return Value
None.

4.3.3. mutex_create

Syntax
```
#include <task.h>
PMUTEX mutex_create(void);
```

Description
Create a new mutex object. Useful when you need synchronized access to shared ressources.

Note
I don’t think the mutual exclusion mofo is useful on Neo Geo, but as the process library is a port of the one I use on Dreamcast, It’s implemented.

Return Value
Pointer to the newly created mutex.

4.3.4. mutex_destroy

Syntax
```
#include <task.h>
void mutex_destroy(PMUTEX mutex);
```

Description
Destroy a mutex object.

Return Value
None.

4.3.5. mutex_release

Syntax
```
#include <task.h>
void mutex_release(PMUTEX mutex);
```

Description
Release a mutex object.

Return Value
None.

4.3.6. mutex_take

Syntax
```
#include <task.h>
void mutex_take(PMUTEX mutex);
```

Description
Take a mutex object.

Return Value
None.

4.3.7. peek_message

Syntax
```
#include <task.h>
DWORD peek_message(PTASK myself, PTASK *from, PDWORD pdata1, PDWORD pdata2);
```

Description
Read a message without clearing it.

*from	Pointer to a pointer of TASK structure. Basically, the task who sent the message
pdata1	Pointer to a dword of data
pdate2	Idem

Note
If you are not interested in one of the parameters, you can pass a NULL pointer.

Return Value

Values	Description
MAILBOX_OK	Message read OK
MAILBOX_NO_MESSAGE	No message

4.3.8. post_message

Syntax
```
#include <task.h>
DWORD post_message(PTASK from, PTASK to, DWORD data1, DWORD data2);
```

Description
Send two dwords of data to another task.
(The other task will know the sender)

Return Value

Values	Description
MAILBOX_OK	Message posted OK
MAILBOX_CANT_POST	There's already a message posted for the destination task. Try again later.

4.3.9. post_message_wait

Syntax
```
#include <task.h>
void post_message_wait(PTASK from, PTASK to, DWORD data1, DWORD data2);
```

Description
Send two dwords of data to another task and wait if necessary.
(The other task will know the sender)

Return Value
None

4.3.10. read_message

Syntax
```
#include <task.h>
DWORD read_message(PTASK myself, PTASK *from, PDWORD pdata1, PDWORD pdata2);
```

Description
Read a message.

*from	Pointer to a pointer of TASK structure. Basically, the task who sent the message
pdata1	Pointer to a dword of data
pdate2	Idem

Note
If you are not interested in one of the parameters, you can pass a NULL pointer.

Return Value

Values	Description
MAILBOX_OK	Message read OK
MAILBOX_NO_MESSAGE	No message

4.3.11. read_message_wait

Syntax
```
#include <task.h>
void read_message_wait(PTASK myself, PTASK *from, PDWORD pdata1, PDWORD pdata2);
```

Description
Read a message. Wait if necessary.

*from	Pointer to a pointer of TASK structure. Basically, the task who sent the message
pdata1	Pointer to a dword of data
pdate2	Idem

Note
If you are not interested in one of the parameters, you can pass a NULL pointer.

Return Value
None.

4.3.12. task_create

Syntax
```
#include <task.h>
PTASK task_create(void *task_addr, DWORD prio, DWORD tag1, DWORD tag2, int nb_args,
...);
```

Description
This function creates a new task pass the specified parameters to it and add it to the task list.
In this library a task is a normal C function. The first parameter is always a pointer to the TASK structure of the task, so the task can manipulate its data itself.

task_addr	Pointer to the task procedure.
prio	Priority for the task. Defined once for all, cannot be changed later. 0x0000 = highest priority, 0xFFFF = lowest priority
tag1, tag2	Freely usable values. You can put whatever you want here.
nb_args	Number of arguments to pass to the task.
..	Arguments to pass.

Return Value
Pointer to the newly created task. NULL on failure.

Note
Task switching is not automatic. You must call _release_timeslice() regularly to pass control to other tasks.

Example

```
…
      task_create(task_bouncing_block, 0x1000, MAKE_ID('B','O','U','N'),
      MAKE_ID('C','E','0','1'), 4, 0, 100, 0, 1);

while(1)
{
      task_exec();
      wait_vbl();
}
…
void task_bouncing_block(PTASK myself, int x, int height, int angle, int step)
{
      …
```

4.3.13. task_enum

Syntax

```
#include <task.h>
void task_enum(DWORD what, DWORD state, DWORD tag1, DWORD tag2, void *user_data,
TASKENUM_CALLBACK callback);
```

Description
This function will call the callback routine for any task matching the specified criterias.

Valid values for what :

Values	Description
TASKENUM_TAG1	Enumerate tasks that match tag1
TASKENUM_TAG2	Enumerate tasks that match tag2
TASKENUM_BOTH	Enumerate tasks that match both tags
TASKENUM_STATE	Enumerate tasks that match the state
TASKENUM_ALL	Enumerate that match all attributes
TASKENUM_NONE	Enumerate all tasks

tag1, tag2, state	see create_task()
*user_data	pointer passed to the callback function. Can be anything you want.
callback	pointer to the callback function, defined as follows : `typedef BOOL (*TASKENUM_CALLBACK)(PTASK task, void *user_data);` The callback function must return TRUE if it wishes to get the next occurrence, or return FALSE to terminate enumeration.

Return Value
None.

4.3.14. task_exec

Syntax
```
#include <task.h>
void task_exec(void);
```

Description
Execute all enabled tasks in the task list.

Return Value
None.

Note
You can call this in an infinite loop or within an interrupt handler.

4.3.15. task_kill

Syntax
```
#include <task.h>
void task_kill(PTASK task);
```

Description
Kill the specified task.

Return Value
None.

Note
If a task kills itself, task_kill automatically calls _release_timeslice()

4.3.16. task_resume

Syntax
```
#include <task.h>
void task_resume(PTASK task);
```

Description
Resumes execution for the specified task.

Return Value
None.

4.3.17. task_sleep

Syntax
```
#include <task.h>
void task_sleep(DWORD ntimes);
```

Description
This will call _release_timeslice() n times. Easy.

Return Value
None.

4.3.18. task_suspend

Syntax
```
#include <task.h>
void task_suspend(PTASK task);
```

Description
Suspends execution for the specified task.

Return Value
None.

5. libvideo

5.1. Predefined types and structures

```
typedef struct {
      WORD              color[16];
} PALETTE, *PPALETTE;
```

A palette of 16 colors. The Neo Geo has 256 palettes of 16 colors = 4096 colors

```
typedef struct {
      WORD              block_number;
      WORD              attributes;
} TILE, *PTILE;
```

A TILE defines the attributes of a 16x16 block. Each sprite is composed of 32 tiles aligned vertically.

```
typedef struct {
      TILE              tiles[32];
} TILEMAP, *PTILEMAP;
```

A TILEMAP is a group of 32 TILEs needed to draw a sprite.

```
typedef struct {
      WORD              r_offset;
      WORD              g_offset;
      WORD              b_offset;
} FADE_TRIPLET, *PFADE_TRIPLET;
```

This is an offset in the fade table. Added to the step (0-31), it gives the value of the corresponding rgb component. (r, g, b)

```
typedef struct {
      FADE_TRIPLET      fade_triplets[16];
} FADER, *PFADER;
```

A group of 16 FADE_TRIPLETs needed to fade a palette.

```
typedef struct {
      BYTE              values[1024];
} FADE_TABLE, *PFADE_TABLE;
```

This is a two dimensional array. X is the step (or time) and Y is the initial value of the (r, g, b) component.

For a fade out we'll have something like this :

0	**1**	**2**	**3**	**4**	**…**	**26**	**27**	**28**	**29**	**30**	**31**
…											
31	31	30	29	28	…	5	4	3	2	1	0

5.2. Functions

5.2.1. change_sprite_pos

Syntax
```
#include <video.h>
void change_sprite_pos(int sprite, int x, int y, int clipping);
```

Description
Change position and clipping size attributes for the specified sprite.

Return Value
None

5.2.2. change_sprite_zoom

Syntax
```
#include <video.h>
void change_sprite_zoom(int sprite, int xz, int yz, int nb);
```

Description
Change reduction factors starting at sprite ‘sprite’ for ‘nb’ sprites.

Return Value
None

5.2.3. clear_fix

Syntax
```
#include <video.h>
void clear_fix(void);
```

Description
Clears fix (text) layer.

Return Value
None

5.2.4. clear_spr

Syntax
```
#include <video.h>
void clear_spr(void);
```

Description
Clears all sprites.

Return Value
None

5.2.5. create_fader

Syntax
```
#include <video.h>
void create_fader(PPALETTE pal, PFADER fader, int nb);
```

Description
Computes ‘nb’ FADER structure(s) for ‘nb’ PALETTEs starting at ‘pal’.

Return Value
None

5.2.6. do_fade

Syntax
```
#include <video.h>
void do_fade(int pal_start, PFADER fader, PFADE_TABLE table, int nb, int step);
```

Description
Do a palette fading for ‘nb’ palettes, using precalculated FADER tables at ‘table’, a color translation table FADE_TABLE for the specified ‘step’ (0-31)

Note
Two color translation tables are included in the video library :
_fade_to_black
_fade_to_white
their names are self-explanatory.

Return Value
None

5.2.7. erase_sprites

Syntax
```
#include <video.h>
void erase_sprites(int nb);
```

Description
Erases ‘nb’ sprites starting at current position

Return Value
None

5.2.8. get_current_sprite

Syntax
```
#include <video.h>
WORD get_current_sprite(void);
```

Description
Get current sprite number.

Return Value
Current sprite number (0-383)

5.2.9. inc_current_sprite

Syntax
```
#include <video.h>
void inc_current_sprite(short increment);
```

Description
Add the specified value to the current sprite number. (The added value can be negative)

Return Value
None

5.2.10. set_current_sprite

Syntax
```
#include <video.h>
void set_current_sprite(WORD npos);
```

Description
Set current sprite.

Return Value
None

5.2.11. set_pal_bank

Syntax
```
#include <video.h>
void set_pal_bank(int palno);
```

Description
Change palette bank. The Neo Geo has two banks of 256 palettes. (0 or 1)

Return Value
None

5.2.12. setpalette

Syntax
```
#include <video.h>
void setpalette(int npal, int nb, const PPALETTE palette);
```

Description
Set ‘nb’ palettes, starting at ‘npal’ using colors definitions stored at ‘palette’.

Return Value
None

5.2.13. textout

Syntax
```
#include <video.h>
void textout(int x, int y, int pal, int bank, const char *txt);
```

Description
Display a string at x,y using specified fix bank and palette.

Return Value
None

5.2.14. textoutf

Syntax
```
#include <video.h>
void textoutf(int x, int y, int pal, int bank, const char *fmt, ...);
```

Description
Similar to textout, but use printf like formatting.

Return Value
None

5.2.15. wait_vbl

Syntax
```
#include <video.h>
void wait_vbl(void);
```

Description
Wait for vertical blank.

Return Value
None

5.2.16. write_sprite_data

Syntax

```
#include <video.h>
WORD  write_sprite_data(int x, int y, int xz, int yz, int clipping, int nb, const
PTILEMAP tilemap);
```

Description

Writes ‘nb’ sprite tilemaps at current position. Also sets attributes (position, reduction factors, clipping). If you write multiple tilemaps (nb > 1) they will be linked using the « sticky bit »

Return Value

None

2. Index