mophun API Reference Manual

Generated by Doxygen 1.3.4

Mon Oct 13 17:11:29 2003

Contents

Chapter 1

mophun API Reference

Version:
1.72

Date:
October 2, 2003.

This manual contains a reference to the mophun API, mophun 3D and mophun Sound.

1.1 Meta info

A meta info tag is a name value pair with text information stored in the game as a resource. The mophun resource compiler is used to compile the text into the game, for information on how to use the resource compiler see the `morc manual`.

Below is a list of standard meta data tags.

Name	Explanation	Maximum recommended length	Usage
Title	The name of the game	60	The name that is shown in the list of games.
Help	A short description of how to play the game, for example keys to use	512	Displayed in an game information.
Copyright info	A copyright string	60	Displayed in an game information.
Vendor	The name of the vendor	60	Displayed in an game information.
Program version	The version number of the game	10	Displayed in an game information.

1.2 Revision history

Revision	Date	Author	Change
1.44	2002-12-13	Harald Walden	Added vStreamMode - description Added vSetBackColor
1.45	2002-12-13	Harald Walden	Added vStreamMode description
1.46	2002-12-13	Harald Walden	Fixed misspellings
1.47	2002-12-13	Harald Walden	Fixed misspellings
1.48	2003-01-24	Harald Walden	Added section about the Data Certificate Library
1.51	2003-02-19	Björn Wennerström	Clarification regarding PPL and time bomb
1.52	2003-02-20	Janne Korpela	Added API Version section
1.53	2003-02-28	Björn Wennerström	Added SONYERICSSON_-T610
1.54	2003-02-28	Björn Wennerström	Added data certificate library section
1.57	2003-04-15	Harald Walden	Fixed misspelled GetButtonData in index
1.58	2003-05-12	Harald Walden	Updated info about clear screen. Fixed misspelled clear screen in index. Added MsgBox and MsgBoxU to index. Added SpriteCollision to index. Added SpriteBoxCollision to index
1.59	2003-05-16	Tomas Johansson	Updated Sound Caps
1.60	2003-05-19	Harald Walden	Updated Index
1.61	2003-05-20	Anders Johansson	Added information about sending and receiving SMS under vStreamOpen
1.62	2003-05-20	Harald Walden	Updated vGetVMGPInfo
1.63	2003-06-26	Harald Walden	Updates due to Flip flags in RTE 1.50: • vSetTransfer-Mode • vDrawTile • vMapInit Added two 2D functions that comes with RTE 1.50: • vGetPixel • vCopyRect

Chapter 2

mophun API Module Index

2.1 mophun API Modules

Here is a list of all modules:

Chapter 3

mophun API Data Structure Index

3.1 mophun API Data Structures

Here are the data structures with brief descriptions:

Chapter 4

mophun API Page Index

4.1 mophun API Related Pages

Here is a list of all related documentation pages:

Chapter 5

mophun API Module Documentation

5.1 HTTP streams

Mophun supports streams that connect to an HTTP server.

Modules

- HTTP methods
- HTTP stream types.

  A user may call vStreamOpen() using one of the HTTP stream types.

- HTTP stream commands.

  A program may call the vStreamFrom() and vStreamTo() functions to perform HTTP specific actions on a stream, for example setting headers or extracting the HTTP status code.

Data Structures

- struct HTTP_HEADERS

  HTTP header struct.

- struct HTTP_RESULT

  HTTP request result.

Defines

- #define STREAM_HTTP_NODATA 0x80000

  HTTP request will not send data.

- #define STREAM_HTTP_METHOD(n) ((n) & STREAM_HTTP_METHOD_MASK)

  Extract HTTP method from mode flags.

Enumerations

- enum THTTPStatusCode {

  HTTPOk = 200, **HTTPCreated** = 201, **HTTPAccepted** = 202, **HTTPNonAuthInfo** = 203,

  HTTPNoContent = 204, **HTTPResetContent** = 205, **HTTPPartialContent** = 206, **HTTPBadRequest** = 400,

  HTTPUnauthorized = 401, **HTTPForbidden** = 403, **HTTPNotFound** = 404, **HTTPMethodNotAllowed** = 405,

  HTTPNotAcceptable = 406, **HTTPProxyAuthRequired** = 407, **HTTPRequestTimeout** = 408, **HTTPConflict** = 409,

  HTTPGone = 410, **HTTPLengthRequired** = 411, **HTTPPreconditionFailed** = 412, **HTTPRequestEntityTooLarge** = 413,

  HTTPRequestURITooLong = 414, **HTTPUnsupportedMediaType** = 415, **HTTPRequestedRangeNotSatisfiable** = 416, **HTTPExpectationFailed** = 417,

  HTTPInternalServerError = 500, **HTTPNotImplemented** = 501, **HTTPBadGateway** = 502, **HTTPServiceUnavailable** = 503,

  HTTPGatewayTimeout = 504, **HTTPVersionNotSupported** = 505 }

  HTTP status codes.

5.1.1 Detailed Description

Mophun supports streams that connect to an HTTP server.

The API supports at least the following HTTP methods:

- GET
- POST

Since:
API version 1.50

HTTP GET example:

```
#include <vmgp.h>
#include <vhttp.h>

int main()
{
        int handle;
        HTTP_RESULT result;
        char *body;

        // Open a stream connection using HTTP GET
        handle = vStreamOpen("http://www.foo.com/cgi-bin/gethiscore.cgi?id=bar", STREAM_HTTP_GET | STREAM_
        if (handle < 0)
                vTerminateVMGP();

        // Get result of request
        result.size = sizeof(result);
        result.method = STREAM_HTTP_RESULT;

        if (vStreamReadFrom(handle, NULL, 0, &result) == 0
                && result.code == HTTPOk
```

```
                && result.contentlength > 0)
        {
                // Get result body
                body = vNewPtr(result.contentlength);
                if (body == NULL)
                {
                        vStreamClose(handle);
                        vTerminateVMGP();
                }

                vStreamRead(handle, body, result.contentlength);

                // Use data...

                vDisposePtr(body);
        }

        vStreamClose(handle);
}
```

HTTP POST example:

```
#include <vmgp.h>
#include <vhttp.h>

int main()
{
        int handle;
        HTTP_RESULT result;
        char *body;

        // Open a stream connection using HTTP POST
        handle = vStreamOpen("http://www.foo.com/cgi-bin/posthiscore.cgi?id=bar?score=1000", STREAM_HTTP_P
        if (handle < 0)
                vTerminateVMGP();

        // Get result of request
        result.size = sizeof(result);
        result.method = STREAM_HTTP_RESULT;

        if (vStreamReadFrom(handle, NULL, 0, &result) == 0
                && result.code == HTTPOk)
        {
                // POST successful
        }

        vStreamClose(handle);
}
```

HTTP POST with body example:

```
#include <vmgp.h>
#include <vhttp.h>

char post_body[] = "This is the body of the post\n";

int main()
{
        int handle;
        HTTP_HEADERS headers;
        HTTP_RESULT result;
        char *body;

        // Open a stream connection using HTTP POST
        handle = vStreamOpen("http://www.foo.com/cgi-bin/postdata.cgi?id=bar", STREAM_HTTP_POST);
```

```
        if (handle < 0)
                vTerminateVMGP();

        headers.size = sizeof(headers);
        headers.method = STREAM_HTTP_HEADERS;
        headers.headers = "Content-Type: text/plain\r\nX-My-Header: Foobar\r\n";

        vStreamWriteTo(handle, post_body, sizeof(post_body) - 1, &headers);

        // Get result of request
        result.size = sizeof(result);
        result.method = STREAM_HTTP_RESULT;

        if (vStreamReadFrom(handle, NULL, 0, &result) == 0
                && result.code == HTTPOk)
        {
                // POST successful
        }

        vStreamClose(handle);
}
```

5.1.2 Define Documentation

5.1.2.1 #define STREAM_HTTP_METHOD(n) ((n) & STREAM_HTTP_METHOD_MASK)

Extract HTTP method from mode flags.

5.1.2.2 #define STREAM_HTTP_NODATA 0x80000

HTTP request will not send data.

Specificy this flag if the HTTP request will not send any data, for example when doing an HTTP GET request.

5.1.3 Enumeration Type Documentation

5.1.3.1 enum THTTPStatusCode

HTTP status codes.

5.2 HTTP methods

Defines

- #define STREAM_HTTP_METHOD_GET 0x00000

  HTTP GET method.

- #define STREAM_HTTP_METHOD_POST 0x10000

  HTTP POST method.

- #define STREAM_HTTP_METHOD_PUT 0x20000

  HTTP PUT method.

- #define STREAM_HTTP_METHOD_HEAD 0x30000

  HTTP HEAD method.

- #define STREAM_HTTP_METHOD_DELETE 0x40000

  HTTP DELETE method.

- #define STREAM_HTTP_METHOD_TRACE 0x50000

  HTTP TRACE method.

- #define STREAM_HTTP_METHOD_OPTIONS 0x60000

  HTTP OPTIONS method.

- #define STREAM_HTTP_METHOD_MASK 0x70000

  HTTP method mask.

5.2.1 Define Documentation

5.2.1.1 #define STREAM_HTTP_METHOD_DELETE 0x40000

HTTP DELETE method.

5.2.1.2 #define STREAM_HTTP_METHOD_GET 0x00000

HTTP GET method.

5.2.1.3 #define STREAM_HTTP_METHOD_HEAD 0x30000

HTTP HEAD method.

5.2.1.4 #define STREAM_HTTP_METHOD_MASK 0x70000

HTTP method mask.

5.2.1.5 #define STREAM_HTTP_METHOD_OPTIONS 0x60000

HTTP OPTIONS method.

5.2.1.6 #define STREAM_HTTP_METHOD_POST 0x10000

HTTP POST method.

5.2.1.7 #define STREAM_HTTP_METHOD_PUT 0x20000

HTTP PUT method.

5.2.1.8 #define STREAM_HTTP_METHOD_TRACE 0x50000

HTTP TRACE method.

5.3 HTTP stream types.

A user may call vStreamOpen() using one of the HTTP stream types.

Defines

- #define STREAM_HTTP_GET (STREAM_HTTP_METHOD_GET|STREAM_HTTP)

  HTTP GET method.

- #define STREAM_HTTP_POST (STREAM_HTTP_METHOD_POST|STREAM_HTTP)

  HTTP POST method.

- #define STREAM_HTTP_PUT (STREAM_HTTP_METHOD_PUT|STREAM_HTTP)

  HTTP PUT method.

- #define STREAM_HTTP_HEAD (STREAM_HTTP_METHOD_HEAD|STREAM_HTTP)

  HTTP HEAD method.

- #define STREAM_HTTP_DELETE (STREAM_HTTP_METHOD_DELETE|STREAM_HTTP)

  HTTP DELETE method.

- #define STREAM_HTTP_TRACE (STREAM_HTTP_METHOD_TRACE|STREAM_HTTP)

  HTTP TRACE method.

- #define STREAM_HTTP_OPTIONS (STREAM_HTTP_METHOD_OPTIONS|STREAM_-HTTP)

  HTTP OPTIONS method.

5.3.1 Detailed Description

A user may call vStreamOpen() using one of the HTTP stream types.

Each HTTP stream type is a combination of the `STREAM_HTTP` stream-type and an HTTP stream, see HTTP methods.

5.3.2 Define Documentation

5.3.2.1 #define STREAM_HTTP_DELETE (STREAM_HTTP_METHOD_DELETE|STREAM_-HTTP)

HTTP DELETE method.

5.3.2.2 #define STREAM_HTTP_GET (STREAM_HTTP_METHOD_GET|STREAM_HTTP)

HTTP GET method.

5.3.2.3 #define STREAM_HTTP_HEAD (STREAM_HTTP_METHOD_HEAD|STREAM_-HTTP)

HTTP HEAD method.

5.3.2.4 #define STREAM_HTTP_OPTIONS (STREAM_HTTP_METHOD_-OPTIONS|STREAM_HTTP)

HTTP OPTIONS method.

5.3.2.5 #define STREAM_HTTP_POST (STREAM_HTTP_METHOD_POST|STREAM_HTTP)

HTTP POST method.

5.3.2.6 #define STREAM_HTTP_PUT (STREAM_HTTP_METHOD_PUT|STREAM_HTTP)

HTTP PUT method.

5.3.2.7 #define STREAM_HTTP_TRACE (STREAM_HTTP_METHOD_TRACE|STREAM_-HTTP)

HTTP TRACE method.

5.4 HTTP stream commands.

A program may call the vStreamFrom() and vStreamTo() functions to perform HTTP specific actions on a stream, for example setting headers or extracting the HTTP status code.

Defines

- #define STREAM_HTTP_HEADERS 0x0000

  Extract or set HTTP headers.

- #define STREAM_HTTP_RESULT 0x0001

  Get HTTP status code, see THTTPStatusCode.

5.4.1 Detailed Description

A program may call the vStreamFrom() and vStreamTo() functions to perform HTTP specific actions on a stream, for example setting headers or extracting the HTTP status code.

5.4.2 Define Documentation

5.4.2.1 #define STREAM_HTTP_HEADERS 0x0000

Extract or set HTTP headers.

5.4.2.2 #define STREAM_HTTP_RESULT 0x0001

Get HTTP status code, see THTTPStatusCode.

5.5 Setting API version

The mophun API has changed slightly in newer versions of the API. To maintain backwards compatibility, the old behaviour is used unless a game explicitly sets the API version that it expects.

To set the API version, the game must be linked with the `-mversion` option. For example, to set the expected API version to 1.30 use the following option when linking the game:

`-mversion=1.30`

See Tilemap and sprites for a list of API differences.

5.6 Integer types

Basic integer types.

Typedefs

- typedef long long int64_t
  64-bit integer type
- typedef unsigned long long uint64_t
  Unsigned 64-bit integer type.
- typedef int int32_t
  32-bit integer type
- typedef unsigned int uint32_t
  Unsigned 32-bit integer type.
- typedef short int16_t
  16-bit integer type
- typedef unsigned short uint16_t
  Unsigned 16-bit integer type.
- typedef char int8_t
  8-bit integer type
- typedef unsigned char uint8_t
  Unsigned 8-bit integer type.
- typedef short fixed16_t
  16-bit fixed point type
- typedef int fixed32_t
  32-bit fixed point type
- typedef unsigned short wchar_t
  Unicode character type.

5.6.1 Detailed Description

Basic integer types.

Mophun defines portable integer types that are compatible with the recent ANSI standard portable integer types.

C integer types.

- char 8-bit

- short 16-bit
- int 32-bit
- long 32-bit
- long long 64-bit

5.6.2 Typedef Documentation

5.6.2.1 typedef short fixed16_t

16-bit fixed point type

5.6.2.2 typedef int fixed32_t

32-bit fixed point type

5.6.2.3 typedef short int16_t

16-bit integer type

5.6.2.4 typedef int int32_t

32-bit integer type

5.6.2.5 typedef long long int64_t

64-bit integer type

5.6.2.6 typedef char int8_t

8-bit integer type

5.6.2.7 typedef unsigned short uint16_t

Unsigned 16-bit integer type.

5.6.2.8 typedef unsigned int uint32_t

Unsigned 32-bit integer type.

5.6.2.9 typedef unsigned long long uint64_t

Unsigned 64-bit integer type.

5.6.2.10 typedef unsigned char uint8_t

Unsigned 8-bit integer type.

5.6.2.11 typedef unsigned short wchar_t

Unicode character type.

5.7 Mophun macros

Defines

- #define VMGP_MAJOR 2

  Major mophun version.

- #define VMGP_MINOR 1

  Minor mophun version.

- #define vOffsetOf(type, member) ((unsigned long)&((type∗)0) → member)

  Get struct member offset.

- #define NULL ((void∗)0)

  A NULL pointer.

- #define memcpy __builtin_memcpy

  Copy bytes.

- #define memmove __builtin_memcpy

  Copy bytes. Handles overlapping source and destination buffers.

- #define memset __builtin_memset

  Fill memory bytes.

5.7.1 Define Documentation

5.7.1.1 #define memcpy __builtin_memcpy

Copy bytes.

Parameters:

dst Destination buffer.

src Source buffer. \

count Number of bytes to copy.

5.7.1.2 #define memmove __builtin_memcpy

Copy bytes. Handles overlapping source and destination buffers.

Parameters:

dst Destination buffer.

src Source buffer.

count Number of bytes to copy.

5.7.1.3 #define memset __builtin_memset

Fill memory bytes.

Parameters:

dst Destination buffer.

int The byte value to store in the destination buffer.

count Number of bytes to store.

5.7.1.4 #define NULL ((void∗)0)

A NULL pointer.

5.7.1.5 #define VMGP_MAJOR 2

Major mophun version.

5.7.1.6 #define VMGP_MINOR 1

Minor mophun version.

5.7.1.7 #define vOffsetOf(type, member) ((unsigned long)&((type∗)0) → member)

Get struct member offset.

Parameters:

type The structure type.

member The name of the member within the structure.

Returns:

The byte offset of the specified member.

5.8 Debugging facilities

Defines

- #define DbgBreak() __asm__ __volatile__ ("break")

  Generate a breakpoint.

- #define _MEMDEBUG

  Define to record record every call to vNewPtr to detect memory leaks when running in the emulator.

- #define NDEBUG

  Define in a final build.

Functions

- void DbgPrintf (const char *fmt,...)

  Print debug message.

5.8.1 Define Documentation

5.8.1.1 #define _MEMDEBUG

Define to record record every call to vNewPtr to detect memory leaks when running in the emulator.

If NDEBUG is defined, _MEMDEBUG is ignored.

5.8.1.2 #define DbgBreak() __asm__ __volatile__ ("break")

Generate a breakpoint.

5.8.1.3 #define NDEBUG

Define in a final build.

If NDEBUG is defined calls to DbgPrintf are removed and _MEMDEBUG is ignored.

5.8.2 Function Documentation

5.8.2.1 void DbgPrintf (const char * *fmt*, ...)

Print debug message.

This function prints a message on the debug console. In a final build, NDEBUG should be defined to remove all debug messages.

Parameters:

fmt A printf-style format string.

5.9 Memory management

Functions

- uint32_t vMemFree (void)
  Get free memory on the heap.
- uint32_t vMaxFreeBlock (void)
  Get largest free memory block on the heap.
- void * vNewPtr (uint32_t size)
  Allocate memory on the heap.
- void * vNewPtrDbg (uint32_t size, const char *file, int line)
  Allocate memory on the heap with debug info.
- void vDisposePtr (void *ptr)
  Release memory.

5.9.1 Function Documentation

5.9.1.1 void vDisposePtr (void * *ptr*)

Release memory.

This function releases memory previously allocated by vNewPtr() or vNewPtrDbg().

Parameters:

ptr A pointer to the memory to release. May be NULL.

5.9.1.2 uint32_t vMaxFreeBlock (void)

Get largest free memory block on the heap.

The value returned is the size of the largest block of memory that may be allocated.

Returns:

The size, in bytes, of the largest free block on the heap.

5.9.1.3 uint32_t vMemFree (void)

Get free memory on the heap.

The value returned is the total number of bytes available on the heap, an allocation request of this amount may not succeed due to fragmentation.

Returns:

The number of free bytes on the heap.

5.9.1.4 void* vNewPtr (uint32_t *size*)

Allocate memory on the heap.

This function allocates memory on the and heap returns a pointer to the allocated memory. The allocated memory is always 4-byte aligned.

Parameters:

size The size of the allocation request. If zero, the result is undefined.

Returns:

A pointer to the allocated memory, NULL if the allocation failed.

5.9.1.5 void* vNewPtrDbg (uint32_t *size*, const char * *file*, int *line*)

Allocate memory on the heap with debug info.

This function is similar to vNewPtr(), but it is only available in the emulator. It records the allocation along with the file and line-number where the function was called. If _MEMDEBUG is defined, vNewPtr() is redefined as vNewPtrDbg() and any calls to vNewPtr() are automatically recorded.

Parameters:

size The size of the allocation request. If zero, the result is undefined.

file Name of the file.

line Line within the file.

Returns:

A pointer to the allocated memory, NULL if the allocation failed.

5.10 Time and date

Data Structures

- struct TIMEDATE

  Time and date.

Functions

- void vGetTimeDate (TIMEDATE *tm)

  Get current date and time.

- void vGetTimeDateUTC (TIMEDATE *tm)

  Get current date and time.

- uint32_t vGetTime (void)

  Get current date and time.

- uint32_t vGetTickCount (void)

  Get milli-second tick count.

5.10.1 Function Documentation

5.10.1.1 uint32_t vGetTickCount (void)

Get milli-second tick count.

This function returns an increasing mill-second tick count. The resolution of the counter and initial value at start of execution is implementation defined.

Returns:
The current tick count.

5.10.1.2 uint32_t vGetTime (void)

Get current date and time.

The returned time is UTC time, expressed as the number of seconds since 1970.

Returns:
The number of seconds since 1970.

Since:
Api version 1.50.

5.10.1.3 void vGetTimeDate (TIMEDATE * *tm*)

Get current date and time.

The returned time is local time.

Parameters:

tm Pointer to a TIMEDATE structure.

5.10.1.4 void vGetTimeDateUTC (TIMEDATE * *tm*)

Get current date and time.

The returned time is UTC time.

Parameters:

tm Pointer to a TIMEDATE structure.

Since:

Api version 1.50.

5.11 Input facilities

Mophun contains functions to get device-independent input for keys such as up/down/left/right and fire.

Modules

- Key codes

  Device independent key codes.

Scan codes

- #define SCANKEY_MASK 0x00FF

  Mask for ASCII code.

- #define SCAN_PROCESSING 0x8000

  If set, keys are being processed.

- #define SCAN_CTRL 0x0100

  The control modifier key is pressed.

- #define SCAN_ALT 0x0200

  The alt modifier key is pressed.

- #define SCAN_SHIFT 0x0400

  The shift modifier key is pressed.

Functions

- uint32_t vGetButtonData (void)

  Get current keys.

- uint32_t vGetPointerPos (void)

  Get current pointer position.

- int32_t vTestKey (uint32_t key)

  Test if a key is pressed.

- int32_t vScanKeys (void)

  Return currently pressed key.

5.11.1 Detailed Description

Mophun contains functions to get device-independent input for keys such as up/down/left/right and fire.

It also has functions to check if a specific key is pressed or which key is currently pressed.

5.11.2 Define Documentation

5.11.2.1 #define SCAN_ALT 0x0200

The alt modifier key is pressed.

5.11.2.2 #define SCAN_CTRL 0x0100

The control modifier key is pressed.

5.11.2.3 #define SCAN_PROCESSING 0x8000

If set, keys are being processed.

5.11.2.4 #define SCAN_SHIFT 0x0400

The shift modifier key is pressed.

5.11.2.5 #define SCANKEY_MASK 0x00FF

Mask for ASCII code.

5.11.3 Function Documentation

5.11.3.1 uint32_t vGetButtonData (void)

Get current keys.

Returns:
A bitmask of currently pressed keys.

See also:
Key codes

5.11.3.2 uint32_t vGetPointerPos (void)

Get current pointer position.

Returns:
The vertical position in the most significant 16 bits, the horizontal position in the least significant 16 bits.

5.11.3.3 int32_t vScanKeys (void)

Return currently pressed key.

Scan keyboard and return an ASCII code in the bottom 8 bits. If no keys are pressed zero is returned. SCANKEY_MASK may be used as a mask to get the ASCII code. If the bit SCAN_PROCESSING is set, keys are currently being processed and the current key is returned in the low 8 bits.

5.11.3.4 int32_t vTestKey (uint32_t *key*)

Test if a key is pressed.

This function returns non-zero if the specified key is pressed. Key-codes in the range 0-127 are ASCII keys, other values are implementation defined.

Parameters:

key The key to check.

Returns:

Non-zero if the key is pressed.

5.12 Key codes

Device independent key codes.

Defines

- #define KEY_UP 0x00000001
  Up.
- #define KEY_DOWN 0x00000002
  Down.
- #define KEY_LEFT 0x00000004
  Left.
- #define KEY_RIGHT 0x00000008
  Right.
- #define KEY_FIRE 0x00000010
  Fire.
- #define KEY_SELECT 0x00000020
  Select.
- #define POINTER_DOWN 0x00000040
  Pointer is pressed.
- #define POINTER_ALTDOWN 0x00000080
  Second pointer-button is pressed.
- #define KEY_FIRE2 0x00000100
  Alternative fire.

5.12.1 Detailed Description

Device independent key codes.

5.12.2 Define Documentation

5.12.2.1 #define KEY_DOWN 0x00000002

Down.

5.12.2.2 #define KEY_FIRE 0x00000010

Fire.

5.12.2.3 #define KEY_FIRE2 0x00000100

Alternative fire.

5.12.2.4 #define KEY_LEFT 0x00000004

Left.

5.12.2.5 #define KEY_RIGHT 0x00000008

Right.

5.12.2.6 #define KEY_SELECT 0x00000020

Select.

Typically used to pause game and back up in menus.

5.12.2.7 #define KEY_UP 0x00000001

Up.

5.12.2.8 #define POINTER_ALTDOWN 0x00000080

Second pointer-button is pressed.

Used on systems with mouse and the second mouse-button is pressed.

5.12.2.9 #define POINTER_DOWN 0x00000040

Pointer is pressed.

Used on systems with a touchscreen or a mouse pointer.

5.13 Message boxes

Modules

- Message box flags

Functions

- int32_t vMsgBox (int32_t flags, const char *msg,...)
  Display a message box.
- int32_t vMsgBoxU (int32_t flags, const pip_wchar_t *msg,...)
  Display a message box.

5.13.1 Function Documentation

5.13.1.1 int32_t vMsgBox (int32_t *flags*, const char * *msg*, ...)

Display a message box.

This function displays a message box waits for the user to select one of the alternatives presented. All text is treated as ASCII.

Parameters:

flags The message box flags. See Message box flags.

msg The message text.

... The message box title, if VMB_TITLE is specified in flags.

Returns:

The users selection. See Message box flags.

5.13.1.2 int32_t vMsgBoxU (int32_t *flags*, const pip_wchar_t * *msg*, ...)

Display a message box.

This function displays a message box waits for the user to select one of the alternatives presented. All text is treated as Unicode.

Parameters:

flags The message box flags. See Message box flags.

msg The message text.

... The message box title, if VMB_TITLE is specified in flags.

Returns:

The users selection. See Message box flags.

5.14 Message box flags

Defines

- #define VMB_SMALL 0
  Make the message box as small as possible.
- #define VMB_BIG 1
  Make the message box large.
- #define VMB_YESNO 2
  Show a yes/no message box.
- #define VMB_OKCANCEL 4
  Show an ok/cancel message box.
- #define VMB_NO 0
  Returned if the user selects no.
- #define VMB_CANCEL 0
  Returned if the user selects cancel.
- #define VMB_OK 1
  Returned if the user selects ok.
- #define VMB_YES 1
  Returned if the user selects yes.
- #define VMB_ERROR 8
  The message box is an error message.
- #define VMB_WARNING 0x10
  The message box is a warning message.
- #define VMB_INFO 0x20
  The message box is an information message.
- #define VMB_QUESTION 0x40
  The message box is a question.
- #define VMB_TITLE 0x80
  The message box has a title.

5.14.1 Define Documentation

5.14.1.1 #define VMB_BIG 1

Make the message box large.

This is only a hint and may be ignored on some platforms.

5.14.1.2 #define VMB_CANCEL 0

Returned if the user selects cancel.

5.14.1.3 #define VMB_ERROR 8

The message box is an error message.

5.14.1.4 #define VMB_INFO 0x20

The message box is an information message.

5.14.1.5 #define VMB_NO 0

Returned if the user selects no.

5.14.1.6 #define VMB_OK 1

Returned if the user selects ok.

5.14.1.7 #define VMB_OKCANCEL 4

Show an ok/cancel message box.

5.14.1.8 #define VMB_QUESTION 0x40

The message box is a question.

5.14.1.9 #define VMB_SMALL 0

Make the message box as small as possible.
This is only a hint and may be ignored on some platforms.

5.14.1.10 #define VMB_TITLE 0x80

The message box has a title.

5.14.1.11 #define VMB_WARNING 0x10

The message box is a warning message.

5.14.1.12 #define VMB_YES 1

Returned if the user selects yes.

5.14.1.13 #define VMB_YESNO 2

Show a yes/no message box.

5.15 Graphics API

Modules

- Transfer modes
- Color macros
- System font sizes

  System font sizes are not defined as a specific size, they are relative.

- System font styles

  System font styles allows text to be displayed using certain styles and effects.

Data Structures

- struct SPRITE

  Sprite header.

- struct VMGPFONT

  Font description.

- struct VMGPPOLY

  Polygon definition.

- struct VMGPRECT

  Rectangle.

- struct VMGPSTRIDEPTR

  Graphics offset description.

Defines

- #define vCharExtentU(ch) vCharExtent((uint16_t)ch)

  Measure character extent.

Functions

- int32_t vWaitVBL (int32_t block)

  Wait for vertical blank.

- void vFlipScreen (int32_t block)

  Update screen.

- void vSetForeColor (int32_t color)

  Set foreground color.

- void vSetBackColor (int32_t color)

Set background color.

- int32_t vSetTransferMode (int32_t mode)

  Set transfer mode.

- void vDrawObject (int16_t x, int16_t y, void *object)

  Draw a SPRITE.

- void vDrawTile (void *data, int32_t format, int16_t x, int16_t y)

  Draw a single tile.

- void vDrawLine (int16_t x0, int16_t y0, int16_t x1, int16_t y1)

  Draw lines.

- void vDrawFlatPolygon (VMGPPOLY *v0)

  Draw a polygon.

- uint16_t vGetPixel (int16_t x, int16_t y)

  Get pixel color.

- void vPlot (int16_t x, int16_t y)

  Draw a pixel.

- void vFillRect (int16_t x0, int16_t y0, int16_t x1, int16_t y1)

  Draw a filled rectangle.

- void vClearScreen (int32_t color)

  Clear screen.

- void vCopyRect (VMGPSTRIDEPTR *dest, VMGPSTRIDEPTR *source, uint16_t width, uint16_t height)

  Copy rectangle in memory.

- void vSetClipWindow (int16_t x0, int16_t y0, int16_t x1, int16_t y1)

  Set clip window.

- void vSetDisplayWindow (uint32_t width, uint32_t height)

  Set display window size.

- void vSetPaletteEntry (uint8_t index, uint32_t rgb)

  Set single palette color.

- uint32_t vGetPaletteEntry (uint8_t index)

  Get palette entry color.

- int32_t vFindRGBIndex (uint32_t rgb)

  Map color value to palette index.

- uint32_t vCreateGrayValue (uint32_t rgb)

  Create grayscale value.

- void vSetPalette (void *pal, uint8_t index, uint16_t count)

  Set whole or portion of palette.

- VMGPFONT * vSetActiveFont (VMGPFONT *pFont)

  Set active font.

- void vPrint (int32_t mode, int32_t x, int32_t y, const char *str)

  Draw text using the current user-defined font.

- uint32_t vFrameTickCount (void)

  Get frame number.

- void vTextOut (int16_t x, int16_t y, const char *str)

  Draw text using current system font.

- void vTextOutU (int16_t x, int16_t y, const pip_wchar_t *str)

  Draw unicode text using current system font.

- uint32_t vTextExtent (const char *str)

  Measure text extents.

- uint32_t vTextExtentU (const pip_wchar_t *str)

  Measure unicode text extents.

- int32_t vCharExtent (uint16_t ch)

  Measure character extent.

- int32_t vSelectFont (int32_t size, int32_t flags, uint16_t ch)

  Select current system font.

5.15.1 Detailed Description

See also:
MapSprite

5.15.2 Define Documentation

5.15.2.1 #define vCharExtentU(ch) vCharExtent((uint16_t)ch)

Measure character extent.

Identical to vCharExtent().

Parameters:
ch The unicode character to measure.

5.15.3 Function Documentation

5.15.3.1 int32_t vCharExtent (uint16_t *ch*)

Measure character extent.

This function measures the extent of the specified character when drawn using the current font. The returned size is in pixels.

Parameters:

ch The character to measure.

Returns:

The width of the text in the least significant 16 bits, the height in the most significant 16 bits.

5.15.3.2 void vClearScreen (int32_t *color*)

Clear screen.

This function clears the whole screen using the specified color. The current clip window is ignored.

Parameters:

color If bit 31 is set, an RGB555 color value, otherwise an index into the current palette.

See also:

vFillRect, vSetForeColor.

5.15.3.3 void vCopyRect (VMGPSTRIDEPTR * *dest*, VMGPSTRIDEPTR * *source*, uint16_t *width*, uint16_t *height*)

Copy rectangle in memory.

Copy a rectangular memory area.

Parameters:

dest Destination memory description.

source Source memory description.

width Width of the area in pixels.

height Height of the area in pixels.

See also:

VMGPSTRIDEPTR, vFillRect, vClearScreen.

Since:

API version 1.50

Remarks:

The following combinations are possible:

- system -> system
- system -> screen
- screen -> screen
- screen -> system

5.15.3.4 uint32_t vCreateGrayValue (uint32_t *rgb*)

Create grayscale value.

This functions finds and returns a luminosity rgb value, the individual RGB elements will be set to (0,0,0) for black and (31,31,31) is white.

Returns:
A packed RGB555 value.

5.15.3.5 void vDrawFlatPolygon (VMGPPOLY * *v0*)

Draw a polygon.

This function draws a flat filled polygon using the current foreground color. The polygon is clipped if necessary.

Parameters:
v0 Pointer to a polygon definition.

See also:
VMGPPOLY, vSetClipWindow, vSetForeColor.

5.15.3.6 void vDrawLine (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*)

Draw lines.

This function draws a line between the specified points (inclusive) using the current foreground color. The line is clipped if necessary.

Parameters:
x0 Horizontal position, start of line.
y0 Vertical position, start of line.
x1 Horizontal position, end of line.
y1 Vertical position, end of line.

See also:
vSetClipWindow, vSetForeColor.

5.15.3.7 void vDrawObject (int16_t *x*, int16_t *y*, void * *object*)

Draw a SPRITE.

This function draws a sprite at the specified position, using the current transfer mode and clip window.

Parameters:
x Horizontal position.
y Vertical position.
object Pointer to a SPRITE structure followed by graphics data.

See also:
SPRITE, vSetTransferMode, vSetClipWindow.

5.15.3.8 void vDrawTile (void * *data*, int32_t *format*, int16_t *x*, int16_t *y*)

Draw a single tile.

Parameters:

data Pointer to tile graphics data.

format The format of the tile data and how to draw the tile. See the following remarks section for details.

x Horizontal position.

y Vertical position.

Remarks:

Bits 0-2 define the pixel format, currently only VCAPS_IND2 to VCAPS_RGB332 are supported. Bit 3 specifies if the tile is transparent. The palette offset (for index formats) is encoded in bits 8-15. Other bits are reserved.

5.15.3.9 void vFillRect (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*)

Draw a filled rectangle.

This function draws a rectangle using the specified coordinates. The rectangle is filled using the current foreground color. This function does not swap the coordinates if x1 is less than x0 or y1 $<$ y0.

Parameters:

x0 Left position.

y0 Top position.

x1 Right position.

y1 Bottom position.

See also:

vSetClipWindow, vSetForeColor, vClearScreen.

5.15.3.10 int32_t vFindRGBIndex (uint32_t *rgb*)

Map color value to palette index.

This function maps an RGB555 color value to the closest matching palette index.

Parameters:

rgb The RGB555 color to match.

Returns:

The closest matching palette index.

See also:

vSetPalette, vSetPaletteEntry, vGetPaletteEntry.

5.15.3.11 void vFlipScreen (int32_t *block*)

Update screen.

Updates the screen with the current contents of the back-buffer to reflect any drawing operations since last call to vFlipScreen.

Parameters:
block If non-zero, the function waits for vertical blank before updating the screen.

Remarks:
As a side-effect of calling vFlipScreen, the current frame number is incremented.

See also:
vFrameTickCount.

5.15.3.12 uint32_t vFrameTickCount (void)

Get frame number.

This function returns the number of frames since the start of execution. This is equal to the number of calls to vFlipScreen().

Returns:
The current frame.

5.15.3.13 uint32_t vGetPaletteEntry (uint8_t *index*)

Get palette entry color.

This function returns the color value at the specified position in the palette.

Parameters:
index THe palette index.

Returns:
The palette color in RGB555 format.

5.15.3.14 uint16_t vGetPixel (int16_t *x*, int16_t *y*)

Get pixel color.

This function returns an RGB555 color value for the pixel at the specified position. If the position is outside the screen the result is undefined.

Parameters:
x Horizontal position.

y Vertical position.

Returns:
The pixel color.

Since:
API version 1.50

See also:
vPlot.

5.15.3.15 void vPlot (int16_t *x*, int16_t *y*)

Draw a pixel.

This function sets a single pixel at the specified position using the current foreground color. The operation is clipped.

Parameters:
x Horizontal position.
y Vertical position.

See also:
vGetPixel, vSetForeColor, vSetClipWindow.

5.15.3.16 void vPrint (int32_t *mode*, int32_t *x*, int32_t *y*, const char * *str*)

Draw text using the current user-defined font.

This function draws a string of text using the font set by a previous call to vSetActiveFont().

Parameters:
mode The transfer mode used when drawing the text, see Transfer modes.
x The starting horizontal position.
y The starting vertical position.
str The text to draw.

See also:
vSetActiveFont, VMGPFONT.

Remarks:
In API version before 1.50 clipped characters were not drawn and rotated fonts were not supported.

5.15.3.17 int32_t vSelectFont (int32_t *size*, int32_t *flags*, uint16_t *ch*)

Select current system font.

This function selects the font used by the following functions:

- vTextOut()
- vTextOutU()
- vCharExtent()
- vCharExtentU()

- vTextExtent()
- vTextExtentU()

Parameters:

size The size of the font, see System font sizes.

flags Font style and effect, see System font styles.

ch A character that should exist in the selected font.

Returns:

The actual flags used.

5.15.3.18 VMGPFONT* vSetActiveFont (VMGPFONT * *pFont*)

Set active font.

This function sets the font used by the vPrint() function.

Parameters:

pFont Pointer to a user-defined font structure.

Returns:

The previous font, NULL if no font was active.

See also:

VMGPFONT, vPrint.

5.15.3.19 void vSetBackColor (int32_t *color*)

Set background color.

This functions sets the current background color. The color may be specified as a palette index or a direct color value.

Example:

```
// Set background color using palette
vSetBackColor(255);
// Set background color directly
vSetBackColor(vRGB(0xff,0,0));
```

Parameters:

color If bit 31 is set, an RGB555 color value, otherwise an index into the current palette.

See also:

vRGB, vSetForeColor

5.15.3.20 void vSetClipWindow (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*)

Set clip window.

This function sets the clip window for most drawing functions. The clip window is inclusive, which means you cannot create a zero-size clip window.

Note that vClearScreen is not affected by the clip window.

Parameters:

x0 Left position of clip window.

y0 Top position of clip window.

x1 Right position of clip window.

y1 Bottom position of clip window.

Remarks:

The maximum size of the clip window is determined by the current display window size, which defaults to the whole screen. The following functions are affect by the clip window:

- vDrawFlatPolygon
- vDrawLine
- vDrawObject
- vDrawTile
- vFillRect
- vPlot
- vPrint
- vTextOut
- vTextOutU
- vUpdateMap
- vUpdateSprite
- vUpdateSpriteMap

5.15.3.21 void vSetDisplayWindow (uint32_t *width*, uint32_t *height*)

Set display window size.

This function sets the size of the screen that is used by a game. Games should use this to inform the runtime about the size of the game display to ensure proper display on platforms with different screen size.

Parameters:

width The desired width of the display.

height The desired height of the display.

See also:

vSetClipWindow.

5.15.3.22 void vSetForeColor (int32_t *color*)

Set foreground color.

This functions sets the current foreground color. The color may be specified as a palette index or a direct color value.

Example:

```
// Set foreground color using palette
vSetForeColor(255);
// Set foreground color directly
vSetForeColor(vRGB(0xff,0,0));
```

Parameters:

color If bit 31 is set, an RGB555 color value, otherwise an index into the current palette.

See also:

vRGB, vSetBackColor

5.15.3.23 void vSetPalette (void * *pal*, uint8_t *index*, uint16_t *count*)

Set whole or portion of palette.

This function updates the current palette starting at the specified palette index. The palette contains RGB555 values.

Parameters:

pal Pointer to array of RGB555 color values.

index The starting index in the palette.

count The number of entries in pal.

See also:

vSetPaletteEntry, vGetPaletteEntry.

5.15.3.24 void vSetPaletteEntry (uint8_t *index*, uint32_t *rgb*)

Set single palette color.

This function sets the color of a single palette entry.

Parameters:

index The palette index.

rgb The RGB555 color value.

See also:

vGetPaletteEntry, vSetPalette, vCreateGrayValue, vRGB.

Remarks:

The implementation may not support display of the exact color specified, in that case the closest color is used.

5.15.3.25 int32_t vSetTransferMode (int32_t *mode*)

Set transfer mode.

See Transfer modes for details.

Parameters:

mode New transfer mode.

Returns:

The previous transfer mode.

5.15.3.26 uint32_t vTextExtent (const char * *str*)

Measure text extents.

This function measures the extents of a specified string when drawn using the current font. The returned size is in pixels.

Parameters:

str The string to measure.

Returns:

The width of the text in the least significant 16 bits, the height in the most significant 16 bits.

5.15.3.27 uint32_t vTextExtentU (const pip_wchar_t * *str*)

Measure unicode text extents.

This function measures the extents of a specified string when drawn using the current font. The returned size is in pixels.

Parameters:

str The string to measure.

Returns:

The width of the text in the least significant 16 bits, the height in the most significant 16 bits.

5.15.3.28 void vTextOut (int16_t *x*, int16_t *y*, const char * *str*)

Draw text using current system font.

This function draws text using the currently selected system font. Fonts are selected by calling vSelectFont().

Parameters:

x Horizontal position of text.

y Vertical position of text.

str The text to draw.

5.15.3.29 void vTextOutU (int16_t *x*, int16_t *y*, const pip_wchar_t * *str*)

Draw unicode text using current system font.

This function draws text using the currently selected system font. Fonts are selected by calling vSelectFont().

Parameters:

x Horizontal position of text.

y Vertical position of text.

str The text to draw.

5.15.3.30 int32_t vWaitVBL (int32_t *block*)

Wait for vertical blank.

This function waits for or checks if the screen is in vertical blank.

Parameters:

block If non-zero the function waits for vertical blank before returning, otherwise it returns immediately.

Returns:

Non-zero if vertical blank was active.

5.16 Transfer modes

Defines

- #define MODE_BLOCK 0

  Block copy mode, no transparency.

- #define MODE_TRANS 1

  Transparent copy mode, color 0 is treated as transparent.

- #define MODE_FLIPX 2

  Flip sprites horizontally.

- #define MODE_FLIPY 4

  Flip sprites vertically.

- #define MODE_ROT90 2

  Text is rotated 90 degrees.

- #define MODE_ROT270 4

  Text is rotated 270 degrees.

5.16.1 Define Documentation

5.16.1.1 #define MODE_BLOCK 0

Block copy mode, no transparency.

5.16.1.2 #define MODE_FLIPX 2

Flip sprites horizontally.

Since:
Api version 1.50.

5.16.1.3 #define MODE_FLIPY 4

Flip sprites vertically.

Since:
Api version 1.50.

5.16.1.4 #define MODE_ROT270 4

Text is rotated 270 degrees.

Only for use with vPrint.

Since:
Api version 1.50.

5.16.1.5 #define MODE_ROT90 2

Text is rotated 90 degrees.

Only for use with vPrint.

Since:
Api version 1.50.

5.16.1.6 #define MODE_TRANS 1

Transparent copy mode, color 0 is treated as transparent.

5.17 Color macros

Defines

- #define vRGB(r, g, b) ((((r) & 0xF8) << 7) | (((g) & 0xF8) << 2) | ((b) >> 3) | 0x80000000)
  Create a color value.
- #define _5TO8(c) (((c)<<3)+4)
  Convert a 5-bit color component to 8 bits.
- #define vRED(rgb) _5TO8(((rgb >> 10) & 0x1f))
  Extract red component from a RGB555 color value.
- #define vGREEN(rgb) _5TO8(((rgb >> 5) & 0x1f))
  Extract green component from a RGB555 color value.
- #define vBLUE(rgb) _5TO8((rgb & 0x1f))
  Extract blue component from a RGB555 color value.
- #define VMGP_WHITE vRGB(0xFF,0xFF,0xFF)
  Predefined white color.
- #define VMGP_BLACK vRGB(0,0,0)
  Predefined black color.
- #define VMGP_GRAY vRGB(0x80,0x80,0x80)
  Predefined gray color.
- #define VMGP_RED vRGB(0xFF,0,0)
  Predefined red color.
- #define VMGP_GREEN vRGB(0,0xFF,0)
  Predefined green color.
- #define VMGP_BLUE vRGB(0,0,0xFF)
  Predefined blue color.
- #define VMGP_YELLOW vRGB(0xFF,0xFF,0)
  Predefined yellow color.
- #define VMGP_MAGENTA vRGB(0xFF,0,0xFF)
  Predefined magenta color.

5.17.1 Define Documentation

5.17.1.1 #define _5TO8(c) (((c)<<3)+4)

Convert a 5-bit color component to 8 bits.

5.17.1.2 #define vBLUE(rgb) _5TO8((rgb & 0x1f))

Extract blue component from a RGB555 color value.

Parameters:
rgb The RGB555 color value.

Returns:
8-bit blue color component.

5.17.1.3 #define vGREEN(rgb) _5TO8(((rgb >> 5) & 0x1f))

Extract green component from a RGB555 color value.

Parameters:
rgb The RGB555 color value.

Returns:
8-bit green color component.

5.17.1.4 #define VMGP_BLACK vRGB(0,0,0)

Predefined black color.

5.17.1.5 #define VMGP_BLUE vRGB(0,0,0xFF)

Predefined blue color.

5.17.1.6 #define VMGP_GRAY vRGB(0x80,0x80,0x80)

Predefined gray color.

5.17.1.7 #define VMGP_GREEN vRGB(0,0xFF,0)

Predefined green color.

5.17.1.8 #define VMGP_MAGENTA vRGB(0xFF,0,0xFF)

Predefined magenta color.

5.17.1.9 #define VMGP_RED vRGB(0xFF,0,0)

Predefined red color.

5.17.1.10 #define VMGP_WHITE vRGB(0xFF,0xFF,0xFF)

Predefined white color.

5.17.1.11 #define VMGP_YELLOW vRGB(0xFF,0xFF,0)

Predefined yellow color.

5.17.1.12 #define vRED(rgb) _5TO8(((rgb >> 10) & 0x1f))

Extract red component from a RGB555 color value.

Parameters:

rgb The RGB555 color value.

Returns:

8-bit red color component.

5.17.1.13 #define vRGB(r, g, b) ((((r) & 0xF8) << 7) | (((g) & 0xF8) << 2) | ((b) >> 3) | 0x80000000)

Create a color value.

Parameters:

r 8-bit red color component.

g 8-bit green color component.

b 8-bit blue color component.

Returns:

An RGB555 color value.

5.18 System font sizes

System font sizes are not defined as a specific size, they are relative.

Defines

- #define FONT_SIZE_NORMAL 0

  Normal system font size.

- #define FONT_SIZE_SMALL 1

  Small system font size.

- #define FONT_SIZE_LARGE 2

  Large system font size.

- #define FONT_SIZE_PIXEL_FLAG 0x80000000UL

  Internal.

- #define FONT_SIZE_POINTS_FLAG 0x40000000UL

  Internal.

- #define FONT_SIZE_PIXELS(n) ((n) | FONT_SIZE_PIXEL_FLAG)

  Specify font height in pixels.

- #define FONT_SIZE_POINTS(n) ((n) | FONT_SIZE_POINTS_FLAG)

  Specify font height in points.

5.18.1 Detailed Description

System font sizes are not defined as a specific size, they are relative.

5.18.2 Define Documentation

5.18.2.1 #define FONT_SIZE_LARGE 2

Large system font size.

5.18.2.2 #define FONT_SIZE_NORMAL 0

Normal system font size.

5.18.2.3 #define FONT_SIZE_PIXEL_FLAG 0x80000000UL

Internal.

5.18.2.4 #define FONT_SIZE_PIXELS(n) ((n) | FONT_SIZE_PIXEL_FLAG)

Specify font height in pixels.

Parameters:
n The font height in pixels.

Returns:
A font size flag.

5.18.2.5 #define FONT_SIZE_POINTS(n) ((n) | FONT_SIZE_POINTS_FLAG)

Specify font height in points.

Parameters:
n The font height in points.

Returns:
A font size flag.

5.18.2.6 #define FONT_SIZE_POINTS_FLAG 0x40000000UL

Internal.

5.18.2.7 #define FONT_SIZE_SMALL 1

Small system font size.

5.19 System font styles

System font styles allows text to be displayed using certain styles and effects.

Defines

- #define FONT_STYLE_NORMAL 0

  Normal style.

- #define FONT_STYLE_ITALIC 1

  Italic style.

- #define FONT_STYLE_BOLD 2

  Bold style.

- #define FONT_STYLE_UNDERLINE 4

  Underlined style.

- #define FONT_STYLE_MONOSPACE 8

  Select monospace font.

- #define **FONT_EFFECT_MASK** 0xF8000000UL
- #define FONT_EFFECT_OUTLINE (1 << 27)

  Draw outline around text.

- #define FONT_EFFECT_SHADOW_LOWERRIGHT (2 << 27)

  Draw shadow below and right of text.

- #define FONT_EFFECT_SHADOW_LOWERLEFT (3 << 27)

  Draw shadow below and left of text.

- #define FONT_EFFECT_SHADOW_UPPERRIGHT (4 << 27)

  Draw shadow above and right of text.

- #define FONT_EFFECT_SHADOW_UPPERLEFT (5 << 27)

  Draw shadow below and left of text.

- #define FONT_EFFECT_SHADOW FONT_EFFECT_SHADOW_LOWERRIGHT

  Alias for FONT_EFFECT_SHADOW_LOWERRIGHT.

5.19.1 Detailed Description

System font styles allows text to be displayed using certain styles and effects.

Font effects are drawn using the current background color.

5.19.2 Define Documentation

5.19.2.1 #define FONT_EFFECT_OUTLINE (1 << 27)

Draw outline around text.

5.19.2.2 #define FONT_EFFECT_SHADOW FONT_EFFECT_SHADOW_LOWERRIGHT

Alias for FONT_EFFECT_SHADOW_LOWERRIGHT.

5.19.2.3 #define FONT_EFFECT_SHADOW_LOWERLEFT (3 << 27)

Draw shadow below and left of text.

5.19.2.4 #define FONT_EFFECT_SHADOW_LOWERRIGHT (2 << 27)

Draw shadow below and right of text.

5.19.2.5 #define FONT_EFFECT_SHADOW_UPPERLEFT (5 << 27)

Draw shadow below and left of text.

5.19.2.6 #define FONT_EFFECT_SHADOW_UPPERRIGHT (4 << 27)

Draw shadow above and right of text.

5.19.2.7 #define FONT_STYLE_BOLD 2

Bold style.

5.19.2.8 #define FONT_STYLE_ITALIC 1

Italic style.

5.19.2.9 #define FONT_STYLE_MONOSPACE 8

Select monospace font.

5.19.2.10 #define FONT_STYLE_NORMAL 0

Normal style.

5.19.2.11 #define FONT_STYLE_UNDERLINE 4

Underlined style.

5.20 System functions

Modules

- System control codes
- Culture identifiers

  The list of possible culture identifiers is defined in vmgplang.h.

Data Structures

- struct COMPRESSEDFILE

  Compressed data header.

Screen orientations

- #define ORIENTATION_PORTRAIT 0

  Portrait screen orientation.

- #define ORIENTATION_LANDSCAPE 1

  Landscape screen orientation.

Defines

- #define VRAND_MAX 0xffff

  The maximum value returned by vGetRandom().

- #define vSetVibrate(on, time) vSysCtl(SYSCTL_SETVIBRATE, on | (time << 16))

  Turn vibrator on/off.

- #define vSetOnScreenInput(on) vSysCtl(SYSCTL_ONSCREENINPUT, on)

  Turn onscreen input on/off.

- #define vSysGetCulture() vSysCtl(SYSCTL_GETCULTURE, 0)

  Get current language.

- #define vSetOrientation(orientation) vSysCtl(SYSCTL_ORIENTATION, orientation)

  Set screen orientation.

Functions

- uint32_t vGetRandom (void)

  Generate a pseudo-random number.

- void vSetRandom (uint32_t seed)

Set seed value for random number generator.

- uint32_t vGetVMGPInfo (void)

  Get mophun version.

- uint32_t vUID (void)

  Get device unique ID.

- void vTerminateVMGP (void)

  Terminate game.

- int vCheckIMEI (const char *imei)

  Check IMEI-number.

- int vCheckNetwork (const char *netstr)

  Check network.

- int vCheckDataCert (const void *cert)

  Verify integrity of a data certificate.

- int vCheckDataCertFile (int32_t handle)

  Verify integrity of a data certificate in a stream.

- uint16_t vSwap16 (uint16_t u16)

  Convert 16-bit value from little-endian to native-endian.

- uint32_t vSwap32 (uint32_t u32)

  Convert 32-bit value from little-endian to native-endian.

- void vSwap (void *ptr, uint32_t n, uint32_t size)

  Convert endian of values in memory.

- int32_t vDecompHdr (COMPRESSEDFILE *hdr, uint8_t *src)

  Get compressed data info.

- int32_t vDecompress (uint8_t *src, uint8_t *dst, int32_t stream, uint32_t ReadBuffSize)

  Decompress data.

- int32_t vSysCtl (int32_t cmd, int32_t opt,...)

  System control function.

5.20.1 Define Documentation

5.20.1.1 #define ORIENTATION_LANDSCAPE 1

Landscape screen orientation.

5.20.1.2 #define ORIENTATION_PORTRAIT 0

Portrait screen orientation.

5.20.1.3 #define VRAND_MAX 0xffff

The maximum value returned by vGetRandom().

5.20.1.4 #define vSetOnScreenInput(on) vSysCtl(SYSCTL_ONSCREENINPUT, on)

Turn onscreen input on/off.

Parameters:
on If non-zero onscreen input controls will be enabled and displayed.

Returns:
Non-zero on success, 0 on failure.

See also:
SYSCTL_ONSCREENINPUT

5.20.1.5 #define vSetOrientation(orientation) vSysCtl(SYSCTL_ORIENTATION, orientation)

Set screen orientation.

Parameters:
orientation Must be ORIENTATION_PORTRAIT or ORIENTATION_LANDSCAPE.

Returns:
Non-zero on success, 0 on failure or the command is not supported.

5.20.1.6 #define vSetVibrate(on, time) vSysCtl(SYSCTL_SETVIBRATE, on | (time << 16))

Turn vibrator on/off.

Parameters:
on If non-zero the vibrator is turned on.

time The time the vibrator should be active, in milliseconds.

Returns:
Non-zero on success, 0 on failure.

See also:
SYSCTL_SETVIBRATE

5.20.1.7 #define vSysGetCulture() vSysCtl(SYSCTL_GETCULTURE, 0)

Get current language.

Returns:
The current language, 0 if the language is undetermined or the command is not supported. See Culture identifiers for a list of supported languages/cultures.

See also:
SYSCTL_GETCULTURE

5.20.2 Function Documentation

5.20.2.1 int vCheckDataCert (const void * *cert*)

Verify integrity of a data certificate.

Parameters:
cert Pointer to data certficate followed by data.

Returns:
Non-zero if certificate was OK, otherwise zero.

See also:
vCheckDataCertFile

5.20.2.2 int vCheckDataCertFile (int32_t *handle*)

Verify integrity of a data certificate in a stream.

This function reads a data certificate and data from a stream and verifies that it is ok.

Parameters:
handle File or resource stream handle.

Returns:
Non-zero if certificate was OK, otherwise zero.

See also:
vCheckDataCert

Since:
API version 1.50

5.20.2.3 int vCheckIMEI (const char * *imei*)

Check IMEI-number.

This function checks an IMEI number matches that of the device.

Parameters:
imei 14-character IMEI number.

Returns:
Non-zero if IMEI number is the same as the devices, otherwise zero.

See also:
vUID.

5.20.2.4 int vCheckNetwork (const char * *netstr*)

Check network.

This function checks if the phone is currently within a specified network.

Parameters:
netstr The network name.

Returns:
Non-zero if the network is the same as the specified network, otherwise zero.

Deprecated
This function has never been implemented.

5.20.2.5 int32_t vDecompHdr (COMPRESSEDFILE * *hdr*, uint8_t * *src*)

Get compressed data info.

This function returns the compressed datas uncompressed size and optionally extracts the header.

Parameters:
hdr Pointer to compressed data header, or NULL if this is not needed.

src Pointer to start of compressed data.

Returns:
The size of the uncompressed data, -1 on error.

5.20.2.6 int32_t vDecompress (uint8_t * *src*, uint8_t * *dst*, int32_t *stream*, uint32_t *ReadBuffSize*)

Decompress data.

Parameters:
src Pointer to compressed data in memory. If decompressing from stream this must be NULL.

dst Pointer to destination buffer.

stream Handle to a file or resource stream, ignored if src is not NULL.

ReadBuffSize This parameter is used when decompressing from a stream. The function manages the allocation and deallocation of this memory. Must be set to a value larger than 80 bytes (more then 1KB is recommended). Ignored if src is not NULL.

Returns:
Size of the decompressed data, -1 on error.

5.20.2.7 uint32_t vGetRandom (void)

Generate a pseudo-random number.

Returns:
A value in the range 0-VRAND_MAX.

See also:
VRAND_MAX.

5.20.2.8 uint32_t vGetVMGPInfo (void)

Get mophun version.

This function returns the version of mophun that the game is executing on.

Returns:
The mophun version, major version is in the 16 most significant bits, minor in the 16 least significant bits.

5.20.2.9 void vSetRandom (uint32_t *seed*)

Set seed value for random number generator.

Using the same seed always generates the same sequence of random numbers, as returned by vGetRandom().

Parameters:
seed The seed for the random number generator.

See also:
vGetRandom, VRAND_MAX.

5.20.2.10 void vSwap (void * *ptr*, uint32_t *n*, uint32_t *size*)

Convert endian of values in memory.

This function converts a block of memory from little-endian to native-endian.

Parameters:
ptr Pointer to start of memory block.

n The number of entries to convert.

size The size of the entries, 2 or 4.

Deprecated
Mophun data is always little-endian.

5.20.2.11 uint16_t vSwap16 (uint16_t *u16*)

Convert 16-bit value from little-endian to native-endian.

Parameters:
u16 The value to convert.

Returns:
The converted value.

Deprecated
Mophun data is always little-endian.

5.20.2.12 uint32_t vSwap32 (uint32_t *u32*)

Convert 32-bit value from little-endian to native-endian.

Parameters:
u32 The value to convert.

Returns:
The converted value.

Deprecated
Mophun data is always little-endian.

5.20.2.13 int32_t vSysCtl (int32_t *cmd*, int32_t *opt*, ...)

System control function.

This function takes a command identifier and an option specific to the command. It is used to perform various system actions.

Parameters:
cmd The command, see System control codes.

opt Option for the command, see System control codes.

Returns:
The return value is defined by the command code.

5.20.2.14 void vTerminateVMGP (void)

Terminate game.

This function terminates the game. All memory allocated by the game is released and all streams are closed. Global destructors are run.

Returns:
This function never returns.

5.20.2.15 uint32_t vUID (void)

Get device unique ID.

This function returns a unique number that identifies the device on which the game is running.

See also:
vCheckIMEI.

5.21 System control codes

Defines

- #define SYSCTL_OFF 0

  Symbol define for false/off.

- #define SYSCTL_ON 1

  Symbol define for true/on.

- #define SYSCTL_SETVIBRATE 0x00000001UL

  Vibrator control.

- #define SYSCTL_ONSCREENINPUT 0x00000002UL

  Onscreen input control.

- #define SYSCTL_ORIENTATION 0x00000003UL

  Screen orientation control.

- #define SYSCTL_GETCULTURE 0x00000004UL

  Get culture.

- #define SYSCTL_VENDOR 0x80000000UL

  Vendor specific command.

5.21.1 Define Documentation

5.21.1.1 #define SYSCTL_GETCULTURE 0x00000004UL

Get culture.

Returns a code for the language the user has selected in the handset. The codes follow the RFC 1766 culture info standard in the format <languagecode>_<country/regioncode>. The low 8 bits specify the <languagecode>, defined by two-letter codes derived from ISO 639-1. The high 8 bits specify the <country/regioncode>, defined by two-letter codes derived from ISO 3166. In cases where a two-letter language code is not available, the three-letter code derived from ISO 639-2 is used; for example, the three-letter code "DIV" is used for cultures that use th Dhivehi language. See Culture identifiers for a list of supported languages/cultures.

5.21.1.2 #define SYSCTL_OFF 0

Symbol define for false/off.

5.21.1.3 #define SYSCTL_ON 1

Symbol define for true/on.

5.21.1.4 #define SYSCTL_ONSCREENINPUT 0x00000002UL

Onscreen input control.

Enable or disable onscreen input controls. The command is ignored on platforms where onscreen input is not supported. Currently only supported on SonyEricsson P800.

5.21.1.5 #define SYSCTL_ORIENTATION 0x00000003UL

Screen orientation control.

On platforms that support both landscape and portrait mode this command is used to change the orientation. The orientation can be ORIENTATION_LANDSCAPE or ORIENTATION_PORTRAIT. The command is ignored on platforms where both modes are not supported.

5.21.1.6 #define SYSCTL_SETVIBRATE 0x00000001UL

Vibrator control.

Control device vibrator. The high 16 bits specify the time in milliseconds during which vibrator is turned on. Bit 0 determine whether vibrator is turned on.

5.21.1.7 #define SYSCTL_VENDOR 0x80000000UL

Vendor specific command.

Vendor/device specific commands start from SYSCTL_VENDOR.

5.22 Culture identifiers

The list of possible culture identifiers is defined in vmgplang.h.

Defines

- #define CULTURE_LANGUAGE(id) ((id) & 0xff)

  Extract main language bits of a culture identifier.

5.22.1 Detailed Description

The list of possible culture identifiers is defined in vmgplang.h.

```
#ifndef VMGPLANG_H
#define VMGPLANG_H

/* Possible return values for SYSCTL_GETCULTURE */
/* CULTURE NAME AND IDENTIFIERS FOR LANGUAGE-COUNTRY/REGION */
#define CULTURE_DEFAULT         0
#define CULTURE_OTHER           0xFFFF
#define CULTURE_INVARIANT       0x007F
#define CULTURE_AF                      0x0036 /* AFRIKAANS */
#define CULTURE_AF_ZA           0x0436 /* AFRIKAANS : SOUTH AFRICA */
#define CULTURE_SQ                      0x001C /* ALBANIAN */
#define CULTURE_SQ_AL           0x041C /* ALBANIAN : ALBANIA */
#define CULTURE_AR                      0x0001 /* ARABIC */
#define CULTURE_AR_DZ           0x1401 /* ARABIC : ALGERIA */
#define CULTURE_AR_BH           0x3C01 /* ARABIC : BAHRAIN */
#define CULTURE_AR_EG           0x0C01 /* ARABIC : EGYPT */
#define CULTURE_AR_IQ           0x0801 /* ARABIC : IRAQ */
#define CULTURE_AR_JO           0x2C01 /* ARABIC : JORDAN */
#define CULTURE_AR_KW           0x3401 /* ARABIC : KUWAIT */
#define CULTURE_AR_LB           0x3001 /* ARABIC : LEBANON */
#define CULTURE_AR_LY           0x1001 /* ARABIC : LIBYA */
#define CULTURE_AR_MA           0x1801 /* ARABIC : MOROCCO */
#define CULTURE_AR_OM           0x2001 /* ARABIC : OMAN */
#define CULTURE_AR_QA           0x4001 /* ARABIC : QATAR */
#define CULTURE_AR_SA           0x0401 /* ARABIC : SAUDI ARABIA */
#define CULTURE_AR_SY           0x2801 /* ARABIC : SYRIA */
#define CULTURE_AR_TN           0x1C01 /* ARABIC : TUNISIA */
#define CULTURE_AR_AE           0x3801 /* ARABIC : UNITED ARAB EMIRATES */
#define CULTURE_AR_YE           0x2401 /* ARABIC : YEMEN */
#define CULTURE_HY                      0x002B /* ARMENIAN */
#define CULTURE_HY_AM           0x042B /* ARMENIAN : ARMENIA */
#define CULTURE_AZ                      0x002C /* AZERI */
#define CULTURE_AZ_AZ_CYRL      0x082C /* AZERI (CYRILLIC) : AZERBAIJAN */
#define CULTURE_AZ_AZ_LATN      0x042C /* AZERI (LATIN) : AZERBAIJAN */
#define CULTURE_EU                      0x002D /* BASQUE */
#define CULTURE_EU_ES           0x042D /* BASQUE : BASQUE */
#define CULTURE_BE                      0x0023 /* BELARUSIAN */
#define CULTURE_BE_BY           0x0423 /* BELARUSIAN : BELARUS */
#define CULTURE_BG                      0x0002 /* BULGARIAN */
#define CULTURE_BG_BG           0x0402 /* BULGARIAN : BULGARIA */
#define CULTURE_CA                      0x0003 /* CATALAN */
#define CULTURE_CA_ES           0x0403 /* CATALAN : CATALAN */
#define CULTURE_ZH              0x0004 /* CHINESE */
#define CULTURE_ZH_HK           0x0C04 /* CHINESE : HONG KONG SAR */
#define CULTURE_ZH_MO           0x1404 /* CHINESE : MACAU SAR */
#define CULTURE_ZH_CN           0x0804 /* CHINESE : CHINA */
#define CULTURE_ZH_CHS          0x0004 /* CHINESE (SIMPLIFIED) */
```

```
#define CULTURE_ZH_SG          0x1004 /* CHINESE : SINGAPORE */
#define CULTURE_ZH_TW          0x0404 /* CHINESE : TAIWAN */
#define CULTURE_ZH_CHT         0x7C04 /* CHINESE (TRADITIONAL) */
#define CULTURE_HR              0x001A /* CROATIAN */
#define CULTURE_HR_HR          0x041A /* CROATIAN : CROATIA */
#define CULTURE_CS              0x0005 /* CZECH */
#define CULTURE_CS_CZ          0x0405 /* CZECH : CZECH REPUBLIC */
#define CULTURE_DA              0x0006 /* DANISH */
#define CULTURE_DA_DK          0x0406 /* DANISH : DENMARK */
#define CULTURE_DIV            0x0065 /* DHIVEHI */
#define CULTURE_DIV_MV         0x0465 /* DHIVEHI : MALDIVES */
#define CULTURE_NL              0x0013 /* DUTCH */
#define CULTURE_NL_BE          0x0813 /* DUTCH : BELGIUM */
#define CULTURE_NL_NL          0x0413 /* DUTCH : THE NETHERLANDS */
#define CULTURE_EN              0x0009 /* ENGLISH */
#define CULTURE_EN_AU          0x0C09 /* ENGLISH : AUSTRALIA */
#define CULTURE_EN_BZ          0x2809 /* ENGLISH : BELIZE */
#define CULTURE_EN_CA          0x1009 /* ENGLISH : CANADA */
#define CULTURE_EN_CB          0x2409 /* ENGLISH : CARIBBEAN */
#define CULTURE_EN_IE          0x1809 /* ENGLISH : IRELAND */
#define CULTURE_EN_JM          0x2009 /* ENGLISH : JAMAICA */
#define CULTURE_EN_NZ          0x1409 /* ENGLISH : NEW ZEALAND */
#define CULTURE_EN_PH          0x3409 /* ENGLISH : PHILIPPINES */
#define CULTURE_EN_ZA          0x1C09 /* ENGLISH : SOUTH AFRICA */
#define CULTURE_EN_TT          0x2C09 /* ENGLISH : TRINIDAD AND TOBAGO */
#define CULTURE_EN_GB          0x0809 /* ENGLISH : UNITED KINGDOM */
#define CULTURE_EN_US          0x0409 /* ENGLISH : UNITED STATES */
#define CULTURE_EN_ZW          0x3009 /* ENGLISH : ZIMBABWE */
#define CULTURE_ET              0x0025 /* ESTONIAN */
#define CULTURE_ET_EE          0x0425 /* ESTONIAN : ESTONIA */
#define CULTURE_FO              0x0038 /* FAROESE */
#define CULTURE_FO_FO          0x0438 /* FAROESE : FAROE ISLANDS */
#define CULTURE_FA              0x0029 /* FARSI / PERSIAN */
#define CULTURE_FA_IR          0x0429 /* FARSI : IRAN */
#define CULTURE_FI              0x000B /* FINNISH */
#define CULTURE_FI_FI          0x040B /* FINNISH : FINLAND */
#define CULTURE_FR              0x000C /* FRENCH */
#define CULTURE_FR_BE          0x080C /* FRENCH : BELGIUM */
#define CULTURE_FR_CA          0x0C0C /* FRENCH : CANADA */
#define CULTURE_FR_FR          0x040C /* FRENCH : FRANCE */
#define CULTURE_FR_LU          0x140C /* FRENCH : LUXEMBOURG */
#define CULTURE_FR_MC          0x180C /* FRENCH : MONACO */
#define CULTURE_FR_CH          0x100C /* FRENCH : SWITZERLAND */
#define CULTURE_GL              0x0056 /* GALICIAN */
#define CULTURE_GL_ES          0x0456 /* GALICIAN : GALICIAN */
#define CULTURE_KA              0x0037 /* GEORGIAN */
#define CULTURE_KA_GE          0x0437 /* GEORGIAN : GEORGIA */
#define CULTURE_DE              0x0007 /* GERMAN */
#define CULTURE_DE_AT          0x0C07 /* GERMAN : AUSTRIA */
#define CULTURE_DE_DE          0x0407 /* GERMAN : GERMANY */
#define CULTURE_DE_LI          0x1407 /* GERMAN : LIECHTENSTEIN */
#define CULTURE_DE_LU          0x1007 /* GERMAN : LUXEMBOURG */
#define CULTURE_DE_CH          0x0807 /* GERMAN : SWITZERLAND */
#define CULTURE_EL              0x0008 /* GREEK */
#define CULTURE_EL_GR          0x0408 /* GREEK : GREECE */
#define CULTURE_GU              0x0047 /* GUJARATI */
#define CULTURE_GU_IN          0x0447 /* GUJARATI : INDIA */
#define CULTURE_HE              0x000D /* HEBREW */
#define CULTURE_HE_IL          0x040D /* HEBREW : ISRAEL */
#define CULTURE_HI              0x0039 /* HINDI */
#define CULTURE_HI_IN          0x0439 /* HINDI : INDIA */
#define CULTURE_HU              0x000E /* HUNGARIAN */
#define CULTURE_HU_HU          0x040E /* HUNGARIAN : HUNGARY */
#define CULTURE_IS              0x000F /* ICELANDIC */
#define CULTURE_IS_IS          0x040F /* ICELANDIC : ICELAND */
#define CULTURE_ID              0x0021 /* INDONESIAN */
#define CULTURE_ID_ID          0x0421 /* INDONESIAN : INDONESIA */
```

```
#define CULTURE_IT                      0x0010 /* ITALIAN */
#define CULTURE_IT_IT           0x0410 /* ITALIAN : ITALY */
#define CULTURE_IT_CH           0x0810 /* ITALIAN : SWITZERLAND */
#define CULTURE_JA                      0x0011 /* JAPANESE */
#define CULTURE_JA_JP           0x0411 /* JAPANESE : JAPAN */
#define CULTURE_KN                      0x004B /* KANNADA */
#define CULTURE_KN_IN           0x044B /* KANNADA : INDIA */
#define CULTURE_KK                      0x003F /* KAZAKH */
#define CULTURE_KK_KZ           0x043F /* KAZAKH : KAZAKHSTAN */
#define CULTURE_KOK             0x0057 /* KONKANI */
#define CULTURE_KOK_IN          0x0457 /* KONKANI : INDIA */
#define CULTURE_KO                      0x0012 /* KOREAN */
#define CULTURE_KO_KR           0x0412 /* KOREAN : KOREA */
#define CULTURE_KY                      0x0040 /* KYRGYZ */
#define CULTURE_KY_KZ           0x0440 /* KYRGYZ : KAZAKHSTAN */
#define CULTURE_LV                      0x0026 /* LATVIAN */
#define CULTURE_LV_LV           0x0426 /* LATVIAN : LATVIA */
#define CULTURE_LT                      0x0027 /* LITHUANIAN */
#define CULTURE_LT_LT           0x0427 /* LITHUANIAN : LITHUANIA */
#define CULTURE_MK                      0x002F /* MACEDONIAN */
#define CULTURE_MK_MK           0x042F /* MACEDONIAN : FYROM */
#define CULTURE_MS                      0x003E /* MALAY */
#define CULTURE_MS_BN           0x083E /* MALAY : BRUNEI */
#define CULTURE_MS_MY           0x043E /* MALAY : MALAYSIA */
#define CULTURE_MR                      0x004E /* MARATHI */
#define CULTURE_MR_IN           0x044E /* MARATHI : INDIA */
#define CULTURE_MN                      0x0050 /* MONGOLIAN */
#define CULTURE_MN_MN           0x0450 /* MONGOLIAN : MONGOLIA */
#define CULTURE_NO                      0x0014 /* NORWEGIAN */
#define CULTURE_NB_NO           0x0414 /* NORWEGIAN : NORWAY */
#define CULTURE_NN_NO           0x0814 /* NORWEGIAN (NYNORSK) : NORWAY */
#define CULTURE_PL                      0x0015 /* POLISH */
#define CULTURE_PL_PL           0x0415 /* POLISH : POLAND */
#define CULTURE_PT                      0x0016 /* PORTUGUESE */
#define CULTURE_PT_BR           0x0416 /* PORTUGUESE : BRAZIL */
#define CULTURE_PT_PT           0x0816 /* PORTUGUESE : PORTUGAL */
#define CULTURE_PA                      0x0046 /* PUNJABI */
#define CULTURE_PA_IN           0x0446 /* PUNJABI : INDIA */
#define CULTURE_RO                      0x0018 /* ROMANIAN */
#define CULTURE_RO_RO           0x0418 /* ROMANIAN : ROMANIA */
#define CULTURE_RU                      0x0019 /* RUSSIAN */
#define CULTURE_RU_RU           0x0419 /* RUSSIAN : RUSSIA */
#define CULTURE_SA                      0x004F /* SANSKRIT */
#define CULTURE_SA_IN           0x044F /* SANSKRIT : INDIA */
#define CULTURE_SR_SP_CYRL      0x0C1A /* SERBIAN (CYRILLIC) : SERBIA */
#define CULTURE_SR_SP_LATN      0x081A /* SERBIAN (LATIN) : SERBIA */
#define CULTURE_SK                      0x001B /* SLOVAK */
#define CULTURE_SK_SK           0x041B /* SLOVAK : SLOVAKIA */
#define CULTURE_SL                      0x0024 /* SLOVENIAN */
#define CULTURE_SL_SI           0x0424 /* SLOVENIAN : SLOVENIA */
#define CULTURE_ES                      0x000A /* SPANISH */
#define CULTURE_ES_AR           0x2C0A /* SPANISH : ARGENTINA */
#define CULTURE_ES_BO           0x400A /* SPANISH : BOLIVIA */
#define CULTURE_ES_CL           0x340A /* SPANISH : CHILE */
#define CULTURE_ES_CO           0x240A /* SPANISH : COLOMBIA */
#define CULTURE_ES_CR           0x140A /* SPANISH : COSTA RICA */
#define CULTURE_ES_DO           0x1C0A /* SPANISH : DOMINICAN REPUBLIC */
#define CULTURE_ES_EC           0x300A /* SPANISH : ECUADOR */
#define CULTURE_ES_SV           0x440A /* SPANISH : EL SALVADOR */
#define CULTURE_ES_GT           0x100A /* SPANISH : GUATEMALA */
#define CULTURE_ES_HN           0x480A /* SPANISH : HONDURAS */
#define CULTURE_ES_MX           0x080A /* SPANISH : MEXICO */
#define CULTURE_ES_NI           0x4C0A /* SPANISH : NICARAGUA */
#define CULTURE_ES_PA           0x180A /* SPANISH : PANAMA */
#define CULTURE_ES_PY           0x3C0A /* SPANISH : PARAGUAY */
#define CULTURE_ES_PE           0x280A /* SPANISH : PERU */
#define CULTURE_ES_PR           0x500A /* SPANISH : PUERTO RICO */
```

```
#define CULTURE_ES_ES           0x0C0A /* SPANISH : SPAIN */
#define CULTURE_ES_UY           0x380A /* SPANISH : URUGUAY */
#define CULTURE_ES_VE           0x200A /* SPANISH : VENEZUELA */
#define CULTURE_SW                      0x0041 /* SWAHILI */
#define CULTURE_SW_KE           0x0441 /* SWAHILI : KENYA */
#define CULTURE_SV                      0x001D /* SWEDISH */
#define CULTURE_SV_FI           0x081D /* SWEDISH : FINLAND */
#define CULTURE_SV_SE           0x041D /* SWEDISH : SWEDEN */
#define CULTURE_SYR             0x005A /* SYRIAC */
#define CULTURE_SYR_SY          0x045A /* SYRIAC : SYRIA */
#define CULTURE_TA                      0x0049 /* TAMIL */
#define CULTURE_TA_IN           0x0449 /* TAMIL : INDIA */
#define CULTURE_TT                      0x0044 /* TATAR */
#define CULTURE_TT_RU           0x0444 /* TATAR : RUSSIA */
#define CULTURE_TE                      0x004A /* TELUGU */
#define CULTURE_TE_IN           0x044A /* TELUGU : INDIA */
#define CULTURE_TH                      0x001E /* THAI */
#define CULTURE_TH_TH           0x041E /* THAI : THAILAND */
#define CULTURE_TR                      0x001F /* TURKISH */
#define CULTURE_TR_TR           0x041F /* TURKISH : TURKEY */
#define CULTURE_UK                      0x0022 /* UKRAINIAN */
#define CULTURE_UK_UA           0x0422 /* UKRAINIAN : UKRAINE */
#define CULTURE_UR                      0x0020 /* URDU */
#define CULTURE_UR_PK           0x0420 /* URDU : PAKISTAN */
#define CULTURE_UZ                      0x0043 /* UZBEK */
#define CULTURE_UZ_UZ_CYRL      0x0843 /* UZBEK (CYRILLIC) : UZBEKISTAN */
#define CULTURE_UZ_UZ_LATN      0x0443 /* UZBEK (LATIN) : UZBEKISTAN */
#define CULTURE_VI                      0x002A /* VIETNAMESE */
#define CULTURE_VI_VN           0x042A /* VIETNAMESE : VIETNAM */

/* ISO 639-1 codes that are missing in RFC 1766. They are given values here instead */
#define CULTURE_AA                      0x00EF /* AFAR */
#define CULTURE_AB                      0x00EE /* ABKHAZIAN */
#define CULTURE_AM                      0x00ED /* AMHARIC */
#define CULTURE_AS                      0x00EC /* ASSAMESE */
#define CULTURE_AY                      0x00EB /* AYMARA */
#define CULTURE_BA                      0x00EA /* BASHKIR */
#define CULTURE_BH                      0x00E9 /* BIHARI */
#define CULTURE_BI                      0x00E8 /* BISLAMA */
#define CULTURE_BN                      0x00E7 /* BENGALI BANGLA */
#define CULTURE_BO                      0x00E6 /* TIBETAN */
#define CULTURE_BR                      0x00E5 /* BRETON */
#define CULTURE_CO                      0x00E4 /* CORSICAN */
#define CULTURE_CY                      0x00E3 /* WELSH */
#define CULTURE_DZ                      0x00E2 /* BHUTANI */
#define CULTURE_EO                      0x00E1 /* ESPERANTO */
#define CULTURE_FJ                      0x00E0 /* FIJI */
#define CULTURE_FY                      0x00DF /* FRISIAN */
#define CULTURE_GA                      0x00DE /* IRISH */
#define CULTURE_GD                      0x00DD /* GAELIC SCOTS GAELIC*/
#define CULTURE_GN                      0x00DC /* GUARANI */
#define CULTURE_HA                      0x00DB /* HAUSA */
#define CULTURE_IA                      0x00DA /* INTERLINGUA */
#define CULTURE_IE                      0x00D9 /* INTERLINGUE */
#define CULTURE_IK                      0x00D8 /* INUPIAK */
//#define CULTURE_IW                    HEBREW - SEE  CULTURE_HE */
#define CULTURE_JI                      0x00D7 /* YIDDISH */
#define CULTURE_JV                      0x00D6 /* JAVANESE */
#define CULTURE_KL                      0x00D5 /* GREENLANDIC */
#define CULTURE_KM                      0x00D4 /* CAMBODIAN (KHMER) */
#define CULTURE_KS                      0x00D3 /* KASHMIRI  */
#define CULTURE_KU                      0x00D2 /* KURDISH */
#define CULTURE_LA                      0x00D1 /* LATIN */
#define CULTURE_LN                      0x00D0 /* LINGALA */
#define CULTURE_LO                      0x00CF /* LAOTHIAN */
#define CULTURE_MG                      0x00CE /* MALAGASY */
#define CULTURE_MI                      0x00CD /* MAORI  */
```

```
#define CULTURE_ML                      0x00CC /* MALAYALAM */
#define CULTURE_MO                      0x00CB /* MOLDAVIAN  */
#define CULTURE_MT                      0x00CA /* MALTESE  */
#define CULTURE_MY                      0x00C9 /* BURMESE  */
#define CULTURE_NA                      0x00C8 /* NAURU  */
#define CULTURE_NE                      0x00C7 /* NEPALI */
#define CULTURE_OC                      0x00C6 /* OCCITAN */
#define CULTURE_OM                      0x00C5 /* OROMO AFAN */
#define CULTURE_OR                      0x00C4 /* ORIYA */
#define CULTURE_PS                      0x00C3 /* PASHTO PUSHTO */
#define CULTURE_QU                      0x00C2 /* QUECHUA */
#define CULTURE_RM                      0x00C1 /* RHAETO-ROMANCE */
#define CULTURE_RN                      0x00C0 /* KIRUNDI */
#define CULTURE_RW                      0x00BF /* KINYARWANDA */
#define CULTURE_SD                      0x00BE /* SINDHI */
#define CULTURE_SG                      0x00BD /* SANGRO */
#define CULTURE_SH                      0x00BC /* SERBO-CROATIAN */
#define CULTURE_SI                      0x00BB /* SINGHALESE */
#define CULTURE_SM                      0x00BA /* SAMOAN */
#define CULTURE_SN                      0x00B9 /* SHONA */
#define CULTURE_SO                      0x00B8 /* SOMALI */
#define CULTURE_SS                      0x00B7 /* SISWATI */
#define CULTURE_ST                      0x00B6 /* SESOTHO */
#define CULTURE_SU                      0x00B5 /* SUDANESE */
#define CULTURE_TG                      0x00B4 /* TAJIK */
#define CULTURE_TI                      0x00B3 /* TIGRINYA */
#define CULTURE_TK                      0x00B2 /* TURKMEN */
#define CULTURE_TL                      0x00B1 /* TAGALOG */
#define CULTURE_TN                      0x00B0 /* SETSWANA */
#define CULTURE_TO                      0x00AF /* TONGA */
#define CULTURE_TS                      0x00AE /* TSONGA */
#define CULTURE_TW                      0x00AD /* TWI */
#define CULTURE_VO                      0x00AC /* VOLAPUK */
#define CULTURE_WO                      0x00AB /* WOLOF */
#define CULTURE_XH                      0x00AA /* XHOSA */
#define CULTURE_YO                      0x00A9 /* YORUBA */
#define CULTURE_ZU                      0x00A8 /* ZULU */

#endif /* VMGPLANG_H */
```

5.22.2 Define Documentation

5.22.2.1 #define CULTURE_LANGUAGE(id) ((id) & 0xff)

Extract main language bits of a culture identifier.

See Culture identifiers for a list culture identifiers.

Parameters:

id The culture identifier as returned by vSysGetCulture().

Returns:

The language identifier.

5.23 Sound functions

Data Structures

- struct BEEP

  Beep sequence element.

Sound types

- #define SOUND_TYPE_BEEP 0

  The sound data contains a beep sequence.

- #define SOUND_TYPE_MIDI 2

  The sound data contains an SMF (Standard Midi File).

- #define SOUND_TYPE_AMR 3

  The sound data contains an AMR file (Adaptive MultiRate).

- #define SOUND_TYPE_MAX 3

  The number of sound types currently supported.

Sound flags

- #define SOUND_FLAG_LOOP 0x0100

  The sound should repeat until stopped by the program.

- #define SOUND_FLAG_STREAM 0x0200

  The sound data should be read from a stream (file or resource).

- #define SOUND_FLAG_STOP 0x0400

  Stop current sound.

Defines

- #define **SOUND_RESOURCE_TYPE_MASK** 0xff

Functions

- void vBeep (uint32_t freq, uint32_t duration)

  Play a tone.

- int32_t vPlayResource (void *data, uint32_t length, uint32_t flags)

  Play a sound sequence.

5.23.1 Define Documentation

5.23.1.1 #define SOUND_FLAG_LOOP 0x0100

The sound should repeat until stopped by the program.

5.23.1.2 #define SOUND_FLAG_STOP 0x0400

Stop current sound.

5.23.1.3 #define SOUND_FLAG_STREAM 0x0200

The sound data should be read from a stream (file or resource).

5.23.1.4 #define SOUND_TYPE_AMR 3

The sound data contains an AMR file (Adaptive MultiRate).

5.23.1.5 #define SOUND_TYPE_BEEP 0

The sound data contains a beep sequence.

5.23.1.6 #define SOUND_TYPE_MAX 3

The number of sound types currently supported.

5.23.1.7 #define SOUND_TYPE_MIDI 2

The sound data contains an SMF (Standard Midi File).

5.23.2 Function Documentation

5.23.2.1 void vBeep (uint32_t *freq*, uint32_t *duration*)

Play a tone.

This function plays a single to at the specified frequency for the specified duration.

Parameters:

freq The tone frequency.

duration The duration of the tone in milli-seconds.

5.23.2.2 int32_t vPlayResource (void * *data*, uint32_t *length*, uint32_t *flags*)

Play a sound sequence.

This function starts playback of a sound sequence. The sound may be a midi-tune, a sequence of beeps (tones) or an AMR (Adaptive MultiRate) encoded sound. The function is also used to stop playback of a sound.

Parameters:

data Pointer to the sound data or a stream handle.

length The length of the sound data.

flags Playback flags and sound type. The sound type must be specified. Specify SOUND_FLAG_-STOP to stop the sound (data and length should be zero).

5.24 Task functions

Mophun supports a form of cooperative multi-threading referred to as tasks.

Defines

- #define vSleep() __asm__ __volatile__ ("sleep" ::: "memory")
  Schedule next task.
- #define vKillTask() __asm__ __volatile__ ("killtask");
  Terminate the current task.

Functions

- int32_t vCreateTask (void *TaskAddr, int32_t p0, int32_t p1, int32_t p2)
  Create a new task.
- void vDisposeTask (int32_t id)
  Terminate specific task.
- int32_t vThisTask (void)
  Get current task id.
- int32_t vTaskAlive (int32_t id)
  Check if task exists.
- int32_t vReceive (void)
  Receive task data.
- int32_t vReceiveAny (int32_t id)
  Receive task data from specific task.
- void vSend (int32_t id, int32_t value)
  Send 32-bit data to a task.
- void vYieldToSystem (void)
  Let system do processing.
- uint32_t vSetStackSize (uint32_t size)
  Set default stack size.

5.24.1 Detailed Description

Mophun supports a form of cooperative multi-threading referred to as tasks.

At startup a program contains a single task which starts executing in the main function. Additional tasks may be created by calling vCreateTask. A task runs until it calls vSleep() or is terminated by a call to vKillTask() or vDisposeTask.

5.24.2 Define Documentation

5.24.2.1 #define vKillTask() __asm__ __volatile__ ("killtask");

Terminate the current task.

This function terminates the current task and schedules a new task for execution. If this is the last task alive the program is terminated, this is however not the preferred way of terminating a program, since global destructors may not be run. Use vTerminateVMGP() to terminate a program, this terminates all tasks.

Returns:
This function never returns.

5.24.2.2 #define vSleep() __asm__ __volatile__ ("sleep" ::: "memory")

Schedule next task.

Calling vSleep puts the calling task to sleep and schedules the next task for execution. If there are no other tasks execution resumes immediately after the call to vSleep.

5.24.3 Function Documentation

5.24.3.1 int32_t vCreateTask (void * *TaskAddr*, int32_t *p0*, int32_t *p1*, int32_t *p2*)

Create a new task.

This function creates a new task. The new task does not start execution immediately, execution starts when all the tasks before it in the task list are either put to sleep by calling vSleep() or terminated by a call to vKillTask() or vDisposeTask().

Parameters:
TaskAddr The task entry point function.

p0 First argument passed to the task function.

p1 Second argument passed to the task function.

p2 Third argument passed to the task function.

Returns:
The identifier of the new task, -1 on failure.

Remarks:
Each task has a separate stack which is allocated on the heap, therefore there must be sufficient free space on the heap for the stack. At startup the default stack size is set to the value specific in the header of the .mpn file. The default may be changed by calling vSetStackSize().

Tasks may communicate with each other by sending 32-bit values to each other.

5.24.3.2 void vDisposeTask (int32_t *id*)

Terminate specific task.

This function terminates a specific task. The task is identified by its task identifier, as returned by vCreate-Task.

Parameters:
id Identifier of task to terminate.

5.24.3.3 int32_t vReceive (void)

Receive task data.

Receive 32-bit value sent to task.

Returns:
32-bit value sent to task from another task.

5.24.3.4 int32_t vReceiveAny (int32_t *id*)

Receive task data from specific task.

Receive 32-bit value from a specific task.

Parameters:
id Identifier of the task to read value from.

Returns:
32-bit value sent to task from another task.

5.24.3.5 void vSend (int32_t *id*, int32_t *value*)

Send 32-bit data to a task.

Parameters:
id Identifier of the task to send data to.

value The data to send.

5.24.3.6 uint32_t vSetStackSize (uint32_t *size*)

Set default stack size.

This function sets the stack size for tasks created by calling vCreateTask().

Parameters:
size The new stack size.

Returns:
The actual stack size used.

5.24.3.7 int32_t vTaskAlive (int32_t *id*)

Check if task exists.

This function checks if a specified task id is valid.

Parameters:
id The task identifier.

Returns:
Non-zero if the task exists.

5.24.3.8 int32_t vThisTask (void)

Get current task id.

Returns:
Identifier of the current task.

5.24.3.9 void vYieldToSystem (void)

Let system do processing.

5.25 Tilemap and sprites

The tilemap and sprite API in mophun may be used to draw whole background maps with sprites.

Data Structures

- struct MAP_HEADER

  Map definition.

Tilemap attributes

- #define MAP_TRANSPARENT 0x1

  The tile map have transparent tiles in the layer.

- #define MAP_USERATTRIBUTE 0x2

  The tile map have an attribute layer.

- #define MAP_AUTOANIM 0x8

  The tile map have auto animated tiles in the layer.

- #define MAP_FLIPX 0x10

  The tile map have horizontal flipped tiles in the layer.

- #define MAP_FLIPY 0x20

  The tile map have vertical flipped tiles in the layer.

Functions

- uint32_t vSpriteInit (uint8_t count)

  Initialize sprite slots.

- void vSpriteSet (uint8_t slot, SPRITE *sprite, int16_t x, int16_t y)

  Set sprite in slot.

- void vSpriteClear (void)

  Clear sprite slots.

- void vSpriteDispose (void)

  Relase sprite slots.

- int16_t vSpriteCollision (uint8_t slot, uint8_t slotfrom, uint8_t slotto)

  Check collision between sprites.

- int16_t vSpriteBoxCollision (const VMGPRECT *box, uint8_t slotfrom, uint8_t slotto)

  Check collision between sprite(s) and a box.

- void vUpdateSprite (void)

  Draw sprite slots.

- void vUpdateSpriteMap (void)

  Draw tilemap and sprite slots.

- uint32_t vMapInit (MAP_HEADER *map)

  Initialize tilemap.

- void vMapSetXY (uint16_t x, uint16_t y)

  Change map x/y coordinate offset.

- void vMapSetTile (uint8_t x, uint8_t y, uint8_t tile)

  Change a background tile on the map.

- void vMapSetAttribute (uint8_t x, uint8_t y, uint8_t attribute)

  Change tile attribute in tilemap.

- uint8_t vMapGetTile (uint8_t x, uint8_t y)

  Get tile in tilemap.

- uint8_t vMapGetAttribute (uint8_t x, uint8_t y)

  Get tile attribute in tilemap.

- uint32_t vMapHeaderUpdate (MAP_HEADER *map)

  Update tilemap definition.

- void vMapDispose (void)

  Dispose current tilemap.

- void vUpdateMap (void)

  Draw current tilemap.

5.25.1 Detailed Description

The tilemap and sprite API in mophun may be used to draw whole background maps with sprites.

This is much more efficient that using separate calls to API functions to draw the different parts of the game screen.

Remarks:

Differecens since API version 1.30:

- If animationspeed is zero in MAP_HEADER, tiles animate every frame, instead of every second frame as in previous versions.
- Tilemaps don't have to fill the whole clipping window to be visible.
- Transparent flag for a layer is now actually used. This means that even
- if a tile has the transparency bit set its drawn as solid if this layer flag is not set.

Differences since API version 1.50:

- Updates to flip flags in vSetTransferMode(), vDrawTile(), vMapInit() and tile engine.

The above changes do not take effect unless the API version is explicitly set in the game. See Setting API version.

5.25.2 Define Documentation

5.25.2.1 #define MAP_AUTOANIM 0x8

The tile map have auto animated tiles in the layer.

5.25.2.2 #define MAP_FLIPX 0x10

The tile map have horizontal flipped tiles in the layer.

5.25.2.3 #define MAP_FLIPY 0x20

The tile map have vertical flipped tiles in the layer.

5.25.2.4 #define MAP_TRANSPARENT 0x1

The tile map have transparent tiles in the layer.

5.25.2.5 #define MAP_USERATTRIBUTE 0x2

The tile map have an attribute layer.

5.25.3 Function Documentation

5.25.3.1 void vMapDispose (void)

Dispose current tilemap.

This function releases all resources used by the current tilemap, as allocated by vMapInit() or vMapHeader-Update().

5.25.3.2 uint8_t vMapGetAttribute (uint8_t *x*, uint8_t *y*)

Get tile attribute in tilemap.

Parameters:

x Horizontal tile position relative to mapstart.

y Vertical tile position relative to mapstart.

Returns:

The specified tiles attributes.

5.25.3.3 uint8_t vMapGetTile (uint8_t *x*, uint8_t *y*)

Get tile in tilemap.

Parameters:

x Horizontal tile position relative to mapstart.

y Vertical tile position relative to mapstart.

Returns:

The tile at the specified position.

5.25.3.4 uint32_t vMapHeaderUpdate (MAP_HEADER * *map*)

Update tilemap definition.

Parameters:

map Pointer to tilemap definition.

Returns:

Non-zero if successful, 0 if request failed.

5.25.3.5 uint32_t vMapInit (MAP_HEADER * *map*)

Initialize tilemap.

This function initializes the map functions with the settings specified in map.

Parameters:

map The tilemap definition.

Returns:

Non-zero if successful, 0 if request failed.

5.25.3.6 void vMapSetAttribute (uint8_t *x*, uint8_t *y*, uint8_t *attribute*)

Change tile attribute in tilemap.

Parameters:

x Horizontal tile position relative to mapstart.

y Vertical tile position relative to mapstart.

attribute New tile attribute value.

5.25.3.7 void vMapSetTile (uint8_t *x*, uint8_t *y*, uint8_t *tile*)

Change a background tile on the map.

Parameters:

x Horizontal tile position relative to mapstart.

y Vertical tile position relative to mapstart.

tile Tile value.

5.25.3.8 void vMapSetXY (uint16_t *x*, uint16_t *y*)

Change map x/y coordinate offset.

This function changes the current tilemaps x/y coordinate offset, used for scrolling.

Parameters:

x Horizontal pixel position relative to mapstart.

y Vertical pixel position relative to mapstart.

5.25.3.9 int16_t vSpriteBoxCollision (const VMGPRECT * *box*, uint8_t *slotfrom*, uint8_t *slotto*)

Check collision between sprite(s) and a box.

This function checks for collisions between sprites and a box. Multiple collisions can be detected by calling the function multiple times and increasing the start slot.

Parameters:

box The box to check collisions against.

slotfrom First sprite slot to check.

slotto Last sprite slot to check.

Returns:

The index of a sprite slot that collides with the specific slot, -1 if no collisions were detected.

5.25.3.10 void vSpriteClear (void)

Clear sprite slots.

This function removes all previously inserted sprites from their sprite slots.

5.25.3.11 int16_t vSpriteCollision (uint8_t *slot*, uint8_t *slotfrom*, uint8_t *slotto*)

Check collision between sprites.

This function checks for collisions between sprites in the specified range of sprite slots. Multiple collisions can be detected by calling the function multiple times and increasing the start slot.

Parameters:

slot Sprite to check collision against.

slotfrom First sprite slot to check.

slotto Last sprite slot to check.

Returns:

The index of a sprite slot that collides with the specific slot, -1 if no collisions were detected.

5.25.3.12 void vSpriteDispose (void)

Relase sprite slots.

This function releases memory allocated by vSpriteInit().

5.25.3.13 uint32_t vSpriteInit (uint8_t *count*)

Initialize sprite slots.

Parameters:

count Number of sprite slots to allocate.

Returns:

Non-zero if successful, 0 on failure.

5.25.3.14 void vSpriteSet (uint8_t *slot*, SPRITE * *sprite*, int16_t *x*, int16_t *y*)

Set sprite in slot.

Insert sprite into selected slot.

Parameters:

slot Sprite slot.

sprite Pointer to sprite to insert.

x Horizontal position.

y Vertical position.

5.25.3.15 void vUpdateMap (void)

Draw current tilemap.

See vUpdateSpriteMap() for instructions regarding clip window settings.

5.25.3.16 void vUpdateSprite (void)

Draw sprite slots.

This function draws all sprites currently inserted in the sprite slot list.

5.25.3.17 void vUpdateSpriteMap (void)

Draw tilemap and sprite slots.

This function draws the current tilemap and all sprites currently inserted in the sprite slot list.

Remarks:

Coordinates provided for vSetClipWindow() is very important when it comes to drawing the background maps. If the clip window settings are wrong the background may not be drawn. If any part of the clip window isn't covered by the map layer, the maplayer may not be drawn.

5.26 String functions

Functions

- int32_t vStrLen (const char *str)

  Get null-terminated string length.

- char * vStrCpy (char *s1, const char *s2)

  Copy null-terminated string.

- char * vStrCat (char *s1, const char *s2)

  Concatenate strings.

- int32_t vStrCmp (const char *s1, const char *s2)

  Compare strings.

- char * vSprintf (char *buf, const char *fmt,...)

  Format string.

- char * vSprintfVa (char *buf, const char *fmt, va_list ap)

  Format string.

- char * vitoa (int32_t val, char *buf, uint8_t len, uint8_t pad)

  Convert number to string.

- char * vutoa (uint32_t val, char *buf, uint8_t len, uint8_t pad)

  Convert number to string.

- int32_t vatoi (const char *str, char **end)

  Convert string to number.

- int32_t vStrLenU (const pip_wchar_t *str)

  Get length of null-terminated unicode string.

- pip_wchar_t * vStrCpyU (pip_wchar_t *s1, const pip_wchar_t *s2)

  Copy unicode string.

- pip_wchar_t * vStrCatU (pip_wchar_t *s1, const pip_wchar_t *s2)

  Concatenate unicode strings.

- int32_t vStrCmpU (const pip_wchar_t *s1, const pip_wchar_t *s2)

  Compare unicode strings.

- pip_wchar_t * vStrToU (pip_wchar_t *s1, const char *s2)

  Convert to unicode string.

5.26.1 Function Documentation

5.26.1.1 int32_t vatoi (const char * *str*, char ** *end*)

Convert string to number.

This function parses a string containing for numbers. The conversion stops at the first non-numeric character. A pointer to the first unconverted character is stored in end if it is not NULL.

Parameters:

str The string to convert.

end Pointer to a string-pointer that is set to point to the first unconverted character. If NULL it is ignored.

Returns:

The converted number.

Since:

API version 1.50.

5.26.1.2 char* vitoa (int32_t *val*, char * *buf*, uint8_t *len*, uint8_t *pad*)

Convert number to string.

This function converts a signed number to a string. The resulting string may be padded to a certain length.

Parameters:

val The number to convert.

buf The destination string.

len The minimum length of the resulting string.

pad The pad character used if the resulting string is less than len.

Returns:

A pointer to the terminating null-character.

See also:

vutoa, vatoi, vSprintf, vSprintfVa.

5.26.1.3 char* vSprintf (char * *buf*, const char * *fmt*, ...)

Format string.

This function is similar to the ANSI sprintf function.

See also:

vSprintfVa.

Since:

API version 1.50.

Remarks:

A format string is a regular null-terminated string containing format escape sequences denoted by a '%'-character. The syntax for the escape sequence is: `%[[0][n]{c,d,e,x,o,b,s,%}`.
The `n` part specifies the field length of the formatted value, for numeric values it may be prefixed by a '0'-character to pad the field with zeroes instead of spaces.

Format codes:

%c	The argument is interpreted as a character.
%d	The argument is interpreted as a signed integer.
%u	The argument is interpreted as an unsigned integer.
%x	The argument is interpreted as an unsigned integer and formatted as a hexadecimal number.
%o	The argument is interpreted as an unsigned integer and formatted as an octal number.
%b	The argument is interpreted as an unsigned integer and formatted as a binary number.
%s	The argument is interpreted as a pointer to a null-terminated string. If the string is NULL, the string "(null)" is inserted.
%%	A literal '%'.

5.26.1.4 char* vSprintfVa (char * *buf*, const char * *fmt*, va_list *ap*)

Format string.

Format a string according to the format specified in fmt. The resulting string is copied to buf. The function is identical to vSprintf(), except that the arguments are supplied in a va_list variable.

Since:

API version 1.50.

5.26.1.5 char* vStrCat (char * *s1*, const char * *s2*)

Concatenate strings.

This function appends the string s2 to the string s1.

Parameters:

s1 The destination string.

s2 The source string.

Returns:

Pointer to the terminating null-character.

See also:

vStrCatU, vStrCpy, vStrCpyU.

Since:

API version 1.50.

5.26.1.6 pip_wchar_t* vStrCatU (pip_wchar_t * *s1*, const pip_wchar_t * *s2*)

Concatenate unicode strings.

This function appends the string s2 to the string s1.

Parameters:

s1 The destination string.

s2 The source string.

Returns:

Pointer to the terminating null-character.

See also:

vStrCat, vStrCpy, vStrCpyU.

Since:

API version 1.50.

5.26.1.7 int32_t vStrCmp (const char * *s1*, const char * *s2*)

Compare strings.

Compare two null-terminated strings. The strings are compared numerically by numeric character code values.

Parameters:

s1 The first string.

s2 The second string.

Returns:

Less than 0 of s1 is considered less than s2, 0 if the strings are identical, higher than 0 if s1 considered higher than s2.

See also:

vStrCmpU.

Since:

API version 1.50.

5.26.1.8 int32_t vStrCmpU (const pip_wchar_t * *s1*, const pip_wchar_t * *s2*)

Compare unicode strings.

Compare two null-terminated unicode strings. The strings are compared numerically by numeric character code values.

Parameters:

s1 The first string.

s2 The second string.

Returns:
Less than 0 of s1 is considered less than s2, 0 if the strings are identical, higher than 0 if s1 considered higher than s2.

See also:
vStrCmp.

Since:
API version 1.50.

5.26.1.9 char* vStrCpy (char * *s1*, const char * *s2*)

Copy null-terminated string.

Parameters:
s1 The destination string.
s2 The string to copy.

Returns:
Pointer to the terminating null-character.

See also:
vStrCpyU, vStrCat, vStrCatU.

5.26.1.10 pip_wchar_t* vStrCpyU (pip_wchar_t * *s1*, const pip_wchar_t * *s2*)

Copy unicode string.

Parameters:
s1 The destination string.
s2 The string to copy.

Returns:
Pointer to the terminating null-character.

See also:
vStrCpy, vStrCat, vStrCatU.

Since:
API version 1.50.

5.26.1.11 int32_t vStrLen (const char * *str*)

Get null-terminated string length.

Parameters:
str The string to measure.

Returns:
Length, in characters, of the specified string, not including '\0'.

See also:
vStrLenU.

5.26.1.12 int32_t vStrLenU (const pip_wchar_t * *str*)

Get length of null-terminated unicode string.

Parameters:
str The string to measure.

Returns:
Length, in characters, of the specified string, not including '\0'.

See also:
vStrLen.

Since:
API version 1.50.

5.26.1.13 pip_wchar_t* vStrToU (pip_wchar_t * *s1*, const char * *s2*)

Convert to unicode string.

This function converts a normal null-terminated string to a null-terminated unicode string. The characters are copied literally.

Parameters:
s1 Destination unicode string.
s2 Source 8-bit character string.

Returns:
Pointer to the terminating null-character.

See also:
vStrCpyU, vStrCatU, vStrCpy, vStrCat.

Since:
API version 1.50.

5.26.1.14 char* vutoa (uint32_t *val*, char * *buf*, uint8_t *len*, uint8_t *pad*)

Convert number to string.

This function converts an unsigned number to a string. The resulting string may be padded to a certain length.

Parameters:
val The number to convert.
buf The destination string.
len The minimum length of the resulting string.
pad The pad character used if the resulting string is less than len.

Returns:
A pointer to the terminating null-character.

See also:
vutoa, vatoi, vSprintf, vSprintfVa.

5.27 mophun 3D API

This section describes the mophun 3D API.

Modules

- Renderstate modes
- 3D Rendering Library

  The 3D rendering library is used to set all 3d render states and to draw polygons on the screen.

- Arithmetic functions
- Matrix functions
- Vector functions

Data Structures

- struct BBOX

  Bounding box.

- struct **BILLBOARD**
- struct DCOLOR

  Diffuse color.

- struct DSCOLOR

  Diffuse and specular color.

- struct **LIGHT**
- struct **LIGHTARGS**
- struct MATRIX

  Matrix type.

- struct PLANE

  Plane.

- struct SCOLOR

  Specular color.

- struct UV

  Texture coordinates.

- struct VECTOR3

  3D vector type.

- struct VECTOR4

  4D vector type.

- struct VERTEX

  Vertex.

- struct **VERTEX_GST**
- struct **VERTEXLIST**

3D enable/disable for on/off states

- #define **VMGP3D_DISABLE** 0
- #define **VMGP3D_ENABLE** 1

Alpha/blend modes on pixels

- #define **VMGP3D_BLEND_ZERO** 1
- #define **VMGP3D_BLEND_ONE** 2
- #define **VMGP3D_BLEND_SRC_ALPHA** 4
- #define **VMGP3D_BLEND_ONE_MINUS_SRC_ALPHA** 8
- #define **VMGP3D_BLEND_ONE_MINUS_COLOR** 16
- #define **VMGP3D_BLEND_COLOR** 32

Alpha/blend modes on texels vs primitives

- #define **VMGP3D_TEXBLEND_DECAL** 1
- #define **VMGP3D_TEXBLEND_MODULATE** 2
- #define **VMGP3D_TEXBLEND_ADD** 4
- #define **VMGP3D_TEXBLEND_REPLACE** 8

Comparison modes

- #define **VMGP3D_NEVER** 1
- #define **VMGP3D_LESS** 2
- #define **VMGP3D_EQUAL** 4
- #define **VMGP3D_LEQUAL** 8
- #define **VMGP3D_GREATER** 16
- #define **VMGP3D_NOTEQUAL** 32
- #define **VMGP3D_GEQUAL** 64
- #define **VMGP3D_ALWAYS** 128

Texture filter modes

- #define **VMGP3D_TEXFILTER_NEAREST** 1
- #define **VMGP3D_TEXFILTER_LINEAR** 2
- #define **VMGP3D_TEXFILTER_MIPNEAREST** 4
- #define **VMGP3D_TEXFILTER_MIPLINEAR** 8
- #define **VMGP3D_TEXFILTER_LINEARMIPNEAREST** 16
- #define **VMGP3D_TEXFILTER_LINEARMIPLINEAR** 32

Functional texture modes

- #define **VMGP3D_TEXTURE_FUNCX** 1
- #define **VMGP3D_TEXTURE_FUNCY** 2
- #define **VMGP3D_TEXTURE_FUNCXY** 3

Cull-modes

- #define **VMGP3D_CULLNONE** 0
- #define **VMGP3D_CULLCW** 1
- #define **VMGP3D_CULLCCW** 2

Shade modes

- #define **VMGP3D_SHADE_FLAT** 1
- #define **VMGP3D_SHADE_GOURAUD** 2

Wrap modes

- #define **VMGP3D_WRAP** 1
- #define **VMGP3D_CLAMP** 2
- #define **VMGP3D_MIRROR** 4

Lighting formula modes

- #define **VMGP3D_LIGHTING_FAST** 0
- #define **VMGP3D_LIGHTING_ACCURATE** 1

Defines

- #define **COLORDESTDIFFUSE** 1
- #define **COLORDESTSPECULAR** 2
- #define **VLIGHT_POINT** 1
- #define **VLIGHT_DIRECTIONAL** 2
- #define **VLIGHT_SPOT** 4
- #define **TRIANGLETYPE_LIST** 0x00
- #define **TRIANGLETYPE_STRIP** 0x10
- #define **TRIANGLETYPE_FAN** 0x20
- #define **LIGHTMATRIX_SET** 0
- #define **LIGHTMATRIX_GET** 1
- #define **LIGHTMATRIX_STATE** 2
- #define **LMS_LOCKED** 0
- #define **LMS_UNLOCKED** 1
- #define **LMS_RESET** 2
- #define **BILLBOARD_CENTER** 0
- #define **BILLBOARD_TOP** 1
- #define **BILLBOARD_BOTTOM** 2
- #define **BILLBOARD_LEFT** 4
- #define **BILLBOARD_RIGHT** 8

Typedefs

- typedef DSCOLOR **MATERIAL**

Functions

- void **vSetViewport** (int32_t left, int32_t top, int32_t width, int32_t height)
- void **vSetAmbientLight** (uint32_t rgb)
- void **vSetLight** (int32_t index, LIGHT *light)
- void **vSetMaterial** (MATERIAL *material)
- void **vResetLights** (void)
- void **vLightPoint** (LIGHTARGS *args)
- int16_t **vRenderPrimitive** (VERTEXLIST *vertexlist, uint16_t format)
- int16_t **vRenderPrimitiveIndexed** (int16_t *indexlist, int16_t indexcount, VERTEXLIST *vertexlist, uint16_t format)
- int16_t **vCollisionPointBox** (VECTOR3 *v, BBOX *box)
- int16_t **vCollisionBoxBox** (BBOX *b1, BBOX *b2)
- int16_t **vCollisionVectorPlane** (VECTOR3 *vd, VECTOR3 line_v[2], PLANE *p)
- int16_t **vCollisionVectorPoly** (VECTOR3 *vd, VECTOR3 line_v[2], VECTOR3 poly_v[3], PLANE *p)
- void **vCreatePlaneFromPoly** (PLANE *p, VECTOR3 poly_v[3])
- int16_t **vBoxInViewFrustum** (BBOX *b)
- void **vDrawBillboard** (BILLBOARD *bb)

5.27.1 Detailed Description

This section describes the mophun 3D API.

Since:
API version 1.50

5.27.2 Definitions

Degrees are always in the range of 0 - 4095, where 4096 equals 360 degrees. See Renderstate modes and valid values for a list of renderstate modes and valid values for each mode.

It is possible to query if the 3D api is present by calling vGetCaps(), see Video capabilities and VCAPS_3D.

5.27.3 Renderstate modes and valid values

Renderstate Modes	Valid values
VMGP3D_FUNCTIONALTEXTUREMODE	VMGP3D_TEXTURE_FUNCX VMGP3D_TEXTURE_FUNCY VMGP3D_TEXTURE_FUNCXY
VMGP3D_SRC_BLEND	VMGP3D_BLEND_ZERO VMGP3D_BLEND_ONE VMGP3D_BLEND_SRC_ALPHA VMGP3D_BLEND_ONE_MINUS_SRC_ALPHA VMGP3D_BLEND_ONE_MINUS_COLOR VMGP3D_BLEND_COLOR
VMGP3D_ZENABLE	VMGP3D_DISABLE VMGP3D_ENABLE
VMGP3D_ZFUNCTION	VMGP3D_NEVER VMGP3D_LESS VMGP3D_EQUAL VMGP3D_LEQUAL VMGP3D_GREATER VMGP3D_NOTEQUAL VMGP3D_GEQUAL VMGP3D_ALWAYS
VMGP3D_CULLMODE	VMGP3D_CULLNONE VMGP3D_CULLCW VMGP3D_CULLCCW
VMGP3D_SHADEMODE	VMGP3D_SHADE_FLAT VMGP3D_SHADE_GOURAUD
VMGP3D_FOGENABLE	VMGP3D_DISABLE VMGP3D_ENABLE
VMGP3D_TEXTUREENABLE	VMGP3D_DISABLE VMGP3D_ENABLE
VMGP3D_PERSPECTIVEENABLE	VMGP3D_DISABLE VMGP3D_ENABLE
VMGP3D_BLENDMODE	VMGP3D_TEXBLEND_DECAL VMGP3D_TEXBLEND_MODULATE VMGP3D_TEXBLEND_ADD VMGP3D_TEXBLEND_REPLACE
VMGP3D_FILTERMODE	VMGP3D_TEXFILTER_NEAREST VMGP3D_TEXFILTER_LINEAR VMGP3D_TEXFILTER_MIPNEAREST VMGP3D_TEXFILTER_MIPLINEAR VMGP3D_TEXFILTER_LINEARMIPNEAREST VMGP3D_TEXFILTER_LINEARMIPLINEAR
VMGP3D_WRAPMODE	VMGP3D_WRAP VMGP3D_CLAMP VMGP3D_MIRROR
VMGP3D_TRANSPARENTENABLE	VMGP3D_DISABLE VMGP3D_ENABLE
VMGP3D_SPECULARENABLE	VMGP3D_DISABLE VMGP3D_ENABLE
VMGP3D_ALPHAENABLE	VMGP3D_DISABLE VMGP3D_ENABLE

5.28 Renderstate modes

Defines

- #define VMGP3D_FUNCTIONALTEXTUREMODE 1
  enable/disable functional texturemapping
- #define VMGP3D_SRC_BLEND 2
  src blendtypes
- #define VMGP3D_DST_BLEND 3
  dst blendtypes
- #define VMGP3D_ZENABLE 4
  z-buffer enable/disable
- #define VMGP3D_ZFUNCTION 5
  specifies z-buffer comparison function
- #define VMGP3D_CULLMODE 6
  culling mode (cw/ccw/none)
- #define VMGP3D_SHADEMODE 7
  flat/gouraud shading
- #define VMGP3D_FOGENABLE 8
  enable fog
- #define VMGP3D_TEXTUREENABLE 9
  enable/disable texturemapping
- #define VMGP3D_PERSPECTIVEENABLE 10
  enable perspective correction
- #define VMGP3D_BLENDMODE 11
  texture blend mode
- #define VMGP3D_ALPHAENABLE 12
  enable/disable alpha channel in vertex
- #define VMGP3D_FILTERMODE 13
  texture filter mode
- #define VMGP3D_WRAPMODE 14
  wrap, mirror or clamp
- #define VMGP3D_TRANSPARENTENABLE 15
  enable transparent in textures(pixel value 0 is transparent)
- #define VMGP3D_SPECULARENABLE 16

enable specular component in vertex

- #define VMGP3D_LIGHTINGENABLE 17
  enable lighing calcualtions in render pipe
- #define VMGP3D_LIGHTINGFORMULA 18
  select lighting formula

5.28.1 Define Documentation

5.28.1.1 #define VMGP3D_ALPHAENABLE 12

enable/disable alpha channel in vertex

5.28.1.2 #define VMGP3D_BLENDMODE 11

texture blend mode

5.28.1.3 #define VMGP3D_CULLMODE 6

culling mode (cw/ccw/none)

5.28.1.4 #define VMGP3D_DST_BLEND 3

dst blendtypes

5.28.1.5 #define VMGP3D_FILTERMODE 13

texture filter mode

5.28.1.6 #define VMGP3D_FOGENABLE 8

enable fog

5.28.1.7 #define VMGP3D_FUNCTIONALTEXTUREMODE 1

enable/disable functional texturemapping

5.28.1.8 #define VMGP3D_LIGHTINGENABLE 17

enable lighing calcualtions in render pipe

5.28.1.9 #define VMGP3D_LIGHTINGFORMULA 18

select lighting formula

5.28.1.10 #define VMGP3D_PERSPECTIVEENABLE 10

enable perspective correction

5.28.1.11 #define VMGP3D_SHADEMODE 7

flat/gouraud shading

5.28.1.12 #define VMGP3D_SPECULARENABLE 16

enable specular component in vertex

5.28.1.13 #define VMGP3D_SRC_BLEND 2

src blendtypes

5.28.1.14 #define VMGP3D_TEXTUREENABLE 9

enable/disable texturemapping

5.28.1.15 #define VMGP3D_TRANSPARENTENABLE 15

enable transparent in textures(pixel value 0 is transparent)

5.28.1.16 #define VMGP3D_WRAPMODE 14

wrap, mirror or clamp

5.28.1.17 #define VMGP3D_ZENABLE 4

z-buffer enable/disable

5.28.1.18 #define VMGP3D_ZFUNCTION 5

specifies z-buffer comparison function

5.29 3D Rendering Library

The 3D rendering library is used to set all 3d render states and to draw polygons on the screen.

Functions

- void vInit3D (void)

  Initialize 3D API.

- uint32_t vSetTexture (void *textureptr, uint32_t format, uint32_t lods, uint32_t count)

  Set texture.

- uint32_t **vSetFunctionalTexture** (void *textureptr, uint32_t size)
- void vDrawPolygon (VERTEX_GST *v1, VERTEX_GST *v2, VERTEX_GST *v3)

  Draw polygon.

- void vSetRenderState (uint32_t mode, uint32_t value)

  Set render state.

- uint32_t **vGetSupportedRenderStates** (uint32_t mode)
- void **vSetFogColor** (uint32_t color)
- uint16_t vGetZBufferValue (uint16_t x, uint16_t y)

  Get z-buffer value at x and y position, will return zero if outside clipwindow.

- void vSetZBuffer (uint16_t value)

  Fill z-buffer with a value, using vSetClipWindow coordinates.

5.29.1 Detailed Description

The 3D rendering library is used to set all 3d render states and to draw polygons on the screen.

5.29.2 Function Documentation

5.29.2.1 void vDrawPolygon (VERTEX_GST * *v1*, VERTEX_GST * *v2*, VERTEX_GST * *v3*)

Draw polygon.

Elements are fetched in the structure depending on the current render states.

Parameters:

v1 Pointer to a VERTEX_GST structure.

v2 Pointer to a VERTEX_GST structure.

v3 Pointer to a VERTEX_GST structure.

5.29.2.2 uint16_t vGetZBufferValue (uint16_t *x*, uint16_t *y*)

Get z-buffer value at x and y position, will return zero if outside clipwindow.

Parameters:

x x position in z-buffer.

y y position in z-buffer.

5.29.2.3 void vInit3D (void)

Initialize 3D API.

Initializes all necessary buffers and states. Must be invoked once before any other 3D render functions are used.

5.29.2.4 void vSetRenderState (uint32_t *mode*, uint32_t *value*)

Set render state.

Parameters:

mode Type of render state to modify.

value Value of the specific render state.

5.29.2.5 uint32_t vSetTexture (void * *textureptr*, uint32_t *format*, uint32_t *lods*, uint32_t *count*)

Set texture.

Parameters:

textureptr Pointer to texture data.

format Graphics format of the texture, currently only RGB332.

lods X-lod is located in the lower byte and y-lod in high byte.

count Number of mipmaps, ignored if mipmapping not supported.

Returns:

Non-zero if successful, 0 on error. **Example**: lods = 0x0404 and count = 2 the top lod is 16x16 and the bottom lod is 4x4.

5.29.2.6 void vSetZBuffer (uint16_t *value*)

Fill z-buffer with a value, using vSetClipWindow coordinates.

Parameters:

value Value to fill z-buffer with.

5.30 Arithmetic functions

Defines

- #define vFDIV(v1, v2) ((fixed32_t)(((int64_t)(v1) << 14) / (v2)))

  Fixed point division.

- #define **vFMUL**(v1, v2) vMul((v1),(v2))
- #define vFTOI(v) ((v) >> 14)

  Truncate fixed point value.

- #define vFIXP(v) ((fixed32_t)((v)*16384))

  Create fixed point value.

- #define vDEG(d) (((0x1000*(d))/360))

  Map degrees.

Functions

- fixed32_t vSin (int32_t deg)

  Calculate sine.

- fixed32_t vCos (int32_t deg)

  Calculate cosine.

- fixed32_t vTan (int32_t deg)

  Calculate tangent.

- fixed32_t vSqrt (fixed32_t val)

  Calculate square-root.

- fixed32_t vPow (fixed32_t val, uint8_t exp)

  Raise to power of exponent.

- fixed32_t vDiv (fixed32_t v1, fixed32_t v2)

  Fixed point division.

- fixed32_t vMul (fixed32_t v1, fixed32_t v2)

  Fixed point multiply.

5.30.1 Define Documentation

5.30.1.1 #define vDEG(d) (((0x1000*(d))/360))

Map degrees.

This function returns a value in the range 0-4096 for the specified number of degrees.

Parameters:
d Degrees, 0-360.

Returns:
Value in the range 0-4096.

5.30.1.2 #define vFDIV(v1, v2) ((fixed32_t)(((int64_t)(v1) << 14) / (v2)))

Fixed point division.

Use vDiv.

5.30.1.3 #define vFIXP(v) ((fixed32_t)((v)*16384))

Create fixed point value.

This function returns a fixed point value from an integer or float.

Parameters:
v The value to convert.

Returns:
Fixed point value.

5.30.1.4 #define vFTOI(v) ((v) >> 14)

Truncate fixed point value.

This function returns the integer part of a fixed point value.

Parameters:
v Fixed point value.

Returns:
Integer part.

5.30.2 Function Documentation

5.30.2.1 fixed32_t vCos (int32_t *deg*)

Calculate cosine.

This function calculates a fixed point cosine value.

Parameters:
deg Degree, range 0 - 4095.

Returns:
Fixed point cosine value.

5.30.2.2 fixed32_t vDiv (fixed32_t *v1*, fixed32_t *v2*)

Fixed point division.

This function performs fixed point division.

Parameters:

v1 Fixed point numerator.

v2 Fixed point denominator.

Returns:

Fixed point result of division.

5.30.2.3 fixed32_t vMul (fixed32_t *v1*, fixed32_t *v2*)

Fixed point multiply.

This function performs fixed point multiplication.

Parameters:

v1 First fixed point value.

v2 Second fixed point value.

Returns:

Fixed point result of multiplication.

5.30.2.4 fixed32_t vPow (fixed32_t *val*, uint8_t *exp*)

Raise to power of exponent.

This function calculates the value of `val` raised to the power of `exp`.

Parameters:

val Fixed point value.

exp Exponent (0-255).

Returns:

`val ^ exp`.

5.30.2.5 fixed32_t vSin (int32_t *deg*)

Calculate sine.

This function calculates a fixed point sine value.

Parameters:

deg Degree, range 0 - 4095.

Returns:

Fixed point sinus value.

5.30.2.6 fixed32_t vSqrt (fixed32_t *val*)

Calculate square-root.

This function calculates a fixed point square root.

Parameters:
val Fixed point value.

Returns:
The fixed-point square root of the argument.

5.30.2.7 fixed32_t vTan (int32_t *deg*)

Calculate tangent.

This function calculates a fixed point tangent.

Parameters:
deg Degree, range 0 - 4095.

Returns:
Fixed point tanget value.

5.31 Matrix functions

Functions

- void vMatrixSetCurrent (MATRIX *m)
  Set current matrix.
- void vMatrixGetCurrent (MATRIX *m)
  Get current matrix.
- void vMatrixIdentity (void)
  Set the current matrix to an identity matrix.
- void vMatrixMultiply (MATRIX *m)
  Multiply current matrix.
- void vMatrixMultiply3x3 (MATRIX *m)
  Multiply rotation components of current matrix.
- void vMatrixTranslate (fixed32_t x, fixed32_t y, fixed32_t z)
  Translate current matrix.
- void vMatrixScale (fixed32_t x, fixed32_t y, fixed32_t z)
  Scale current matrix.
- void vMatrixRotateX (int32_t d)
  Rotate current matrix around x axis.
- void vMatrixRotateY (int32_t d)
  Rotate current matrix around y axis.
- void vMatrixRotateZ (int32_t d)
  Rotate current matrix around z axis.
- void vMatrixRotateVector (VECTOR3 *v, int32_t d)
  Rotates current matrix around arbitrary axis.
- void vMatrixTranspose (void)
  Transpose current matrix.
- void vMatrixInvert (void)
  Invert current matrix.
- void vMatrixLookAt (VECTOR3 *veye, VECTOR3 *vat, VECTOR3 *vup)
  Create viewing matrix.
- void vMatrixPerspective (fixed32_t width, fixed32_t height, fixed32_t znear, fixed32_t zfar)
  Replace current matrix with perspective matrix.
- void vMatrixSetLight (MATRIX *m)

Set light matrix.

- void vMatrixSetProjection (MATRIX *m)

  Set projection matrix.

5.31.1 Function Documentation

5.31.1.1 void vMatrixGetCurrent (MATRIX * *m*)

Get current matrix.

This function copies current matrix to the specified matrix.

Parameters:

m Pointer to matrix to receive current matrix.

5.31.1.2 void vMatrixIdentity (void)

Set the current matrix to an identity matrix.

This function sets the current matrix to an identity matrix.

5.31.1.3 void vMatrixInvert (void)

Invert current matrix.

This function inverts the current matrix.

5.31.1.4 void vMatrixLookAt (VECTOR3 * *veye*, VECTOR3 * *vat*, VECTOR3 * *vup*)

Create viewing matrix.

This function creates a viewing matrix that is multiplied with the current matrix.

Parameters:

veye Pointer to a 3D vector that specifies the eye point.

vat Pointer to a 3D vector that specifies the look at (reference) point.

vup Pointer to a 3D vector that specifies the direction of the up vector.

5.31.1.5 void vMatrixMultiply (MATRIX * *m*)

Multiply current matrix.

This function multiplies the current matrix with the specified matrix.

Parameters:

m Matrix to multiply.

5.31.1.6 void vMatrixMultiply3x3 (MATRIX * *m*)

Multiply rotation components of current matrix.

This function multiplies the rotation components of the current matrix with the specified matrix.

Parameters:

m Matrix to multiply.

5.31.1.7 void vMatrixPerspective (fixed32_t *width*, fixed32_t *height*, fixed32_t *znear*, fixed32_t *zfar*)

Replace current matrix with perspective matrix.

This function replaces the current matrix with a perspective matrix.

Parameters:

width Specifies the width of the view volume at the near plane in 3D coordinate space.

height Specifies the height of the view volume at the near plane in 3D coordinate space.

znear Specifies the nearplane in 3D coordinate space.

zfar Specifies the farplane in 3D coordinate space.

Example:

```
// Creates a perspective matrix and then sets the projection matrix
// to that matrix. Note that the height is divided with the width
// to get correct aspect ratio.
vMatrixPerspective(vFIXP(2.0), vFDIV(h << 14, w << 14) * 2, vFIXP(3.0f), vFIXP(100.0f));
vMatrixSetProjection(NULL);
```

5.31.1.8 void vMatrixRotateVector (VECTOR3 * *v*, int32_t *d*)

Rotates current matrix around arbitrary axis.

This function rotates the current matrix with d degrees around an arbitrary axis.

Parameters:

v Pointer to a normalized 3D vector to rotate around.

d Degrees to rotate, in the range 0 - 4095.

5.31.1.9 void vMatrixRotateX (int32_t *d*)

Rotate current matrix around x axis.

This function rotates the current matrix d degrees around the x axis.

Parameters:

d Degrees to rotate, in the range 0 - 4095.

5.31.1.10 void vMatrixRotateY (int32_t *d*)

Rotate current matrix around y axis.

This function rotates the current matrix around the y axis.

Parameters:

d Degrees to rotate, in the range 0 - 4095.

5.31.1.11 void vMatrixRotateZ (int32_t *d*)

Rotate current matrix around z axis.

This function rotates the current matrix around the z axis.

Parameters:

d Degrees to rotate, in the range 0 - 4095.

5.31.1.12 void vMatrixScale (fixed32_t *x*, fixed32_t *y*, fixed32_t *z*)

Scale current matrix.

This function scales the current matrix with x, y and z.

Parameters:

x Scale along the x axis.

y Scale along the y axis.

z Scale along the z axis.

5.31.1.13 void vMatrixSetCurrent (MATRIX * *m*)

Set current matrix.

This function copies the specified matrix to the current matrix.

Parameters:

m New matrix.

5.31.1.14 void vMatrixSetLight (MATRIX * *m*)

Set light matrix.

The light matrix is used for lighting calculations within the T&L render pipe if lighting is enabled. For lighting to work correctly the light matrix should be the inverse of the world matrix (which includes all matrix operations in object and world space).

Parameters:

m Pointer to a matrix. If m is NULL the current matrix is set as the light matrix.

5.31.1.15 void vMatrixSetProjection (MATRIX * *m*)

Set projection matrix.

The current matrix is multiplied with the projection matrix before transformation is done in the T&L render pipe, vVectorTransformV4() and vDrawBillboard() functions.

Parameters:

m Pointer to a matrix. If m is NULL the current matrix is set as the projection matrix.

5.31.1.16 void vMatrixTranslate (fixed32_t *x*, fixed32_t *y*, fixed32_t *z*)

Translate current matrix.

This function translates the current matrix with x, y and z.

Parameters:

x Translation along the x axis.

y Translation along the y axis.

z Translation along the z axis.

5.31.1.17 void vMatrixTranspose (void)

Transpose current matrix.

This function transposes the current matrix.

5.32 Vector functions

Functions

- fixed32_t vDotProduct (VECTOR3 *v1, VECTOR3 *v2)
  Calculate dot product.
- void vCrossProduct (VECTOR3 *vd, VECTOR3 *vs1, VECTOR3 *vs2)
  Calculate cross product.
- void vVectorNormalize (VECTOR3 *v)
  Normalize vector.
- void vVectorAdd (VECTOR3 *d, VECTOR3 *s1, VECTOR3 *s2)
  Add vectors.
- void vVectorArrayAdd (VECTOR3 *d, VECTOR3 *s1, VECTOR3 *s2, int32_t count)
  Add vector arrays.
- void vVectorSub (VECTOR3 *d, VECTOR3 *s1, VECTOR3 *s2)
  Subtract vectors.
- void vVectorMul (VECTOR3 *d, VECTOR3 *s1, VECTOR3 *s2)
  Multiply vectors.
- void vVectorArrayDelta (VECTOR3 *d, VECTOR3 *s1, VECTOR3 *s2, uint32_t param)
  Calculate delta vectors.
- void vVectorTransformV3 (VECTOR3 *vd, VECTOR3 *vs, int32_t count)
  Transform vector array with current matrix.
- void vVectorTransformV4 (VECTOR4 *vd, VECTOR3 *vs, int32_t count)
  Transform vector array with current matrix.
- void vVectorProjectV3 (VECTOR4 *vd, VECTOR3 *vs, int32_t count)
  Transform and project vector array with projection matrix and viewport.
- void vVectorProjectV4 (VECTOR4 *vd, VECTOR4 *vs, int32_t count)
  Project and scale vector array with viewport.

5.32.1 Function Documentation

5.32.1.1 void vCrossProduct (VECTOR3 * *vd*, VECTOR3 * *vs1*, VECTOR3 * *vs2*)

Calculate cross product.

This function calculates the cross product of vectors vs1 and vs2 and stores the result in vector vd.

Parameters:

vd Pointer to destination vector.

vs1 Pointer to first source vector.

vs2 Pointer to second source vector.

5.32.1.2 fixed32_t vDotProduct (VECTOR3 * *v1*, VECTOR3 * *v2*)

Calculate dot product.

This function calculates the dot product of v1 and v2 and returns it as a fixed point value.

Parameters:

v1 Pointer to first vector.

v2 Pointer to second vector.

Returns:

The dot product as a fixed point value.

5.32.1.3 void vVectorAdd (VECTOR3 * *d*, VECTOR3 * *s1*, VECTOR3 * *s2*)

Add vectors.

This function adds two vectors and stores the result in the specified vector.

Parameters:

d Pointer to destination vector.

s1 Pointer to first source vector.

s2 Pointer to second source vector.

5.32.1.4 void vVectorArrayAdd (VECTOR3 * *d*, VECTOR3 * *s1*, VECTOR3 * *s2*, int32_t *count*)

Add vector arrays.

This function adds two arrays of vectors and stores the result in the specified vector.

Parameters:

d Pointer to destination vector.

s1 Pointer to first source vector array.

s2 Pointer to second source vector array.

count The number of vectors to add.

5.32.1.5 void vVectorArrayDelta (VECTOR3 * *d*, VECTOR3 * *s1*, VECTOR3 * *s2*, uint32_t *param*)

Calculate delta vectors.

This function calculates delta vectors divided into a variable number of steps. Divides the delta between vectors in array s1 and vectors in array s2 with steps (see parameters) and stores the result in vector array d.

Parameters:

d Pointer to destination vector array.

s1 Pointer to source vector array 1.

s2 Pointer to source vector array 2.

param Low 16 bits is the number of vectors to traverse. High 16 bits is the number of steps to divide with.

Remarks:

Formula: d = (s2 - s1) / (param >> 16).

5.32.1.6 void vVectorMul (VECTOR3 * *d*, VECTOR3 * *s1*, VECTOR3 * *s2*)

Multiply vectors.

This function multiplies vectors s1 and s2 and stores the result in vector vd.

Parameters:

d Pointer to destination vector.

s1 Pointer to first source vector.

s2 Pointer to second source vector.

5.32.1.7 void vVectorNormalize (VECTOR3 * *v*)

Normalize vector.

This function normalizes the specified vector.

Parameters:

v The vector to normalize.

5.32.1.8 void vVectorProjectV3 (VECTOR4 * *vd*, VECTOR3 * *vs*, int32_t *count*)

Transform and project vector array with projection matrix and viewport.

This function transforms vectors in array vs with the projection matrix and then projects and scales them to the viewport with the settings set with vSetViewport(). The resulting vectors are stored in vector array vd.

Parameters:

vd Pointer to destination 4D vector array.

vs Pointer to source 3D vector array.

count Number of vectors to project.

5.32.1.9 void vVectorProjectV4 (VECTOR4 * *vd*, VECTOR4 * *vs*, int32_t *count*)

Project and scale vector array with viewport.

This function projects and scales vectors in vector array vs to the viewport with the settings set with vSetViewport() and stores the resulting vectors in vector array vd.

Parameters:

vd Pointer to destination 4D vector array.

vs Pointer to source 4D vector array.

count Number of vectors to project.

5.32.1.10 void vVectorSub (VECTOR3 * *d*, VECTOR3 * *s1*, VECTOR3 * *s2*)

Subtract vectors.

This function subtracts vector s2 from vector s1 and stores the result in vector vd.

Parameters:

d Pointer to destination vector.

s1 Pointer to first source vector.

s2 Pointer to second source vector.

5.32.1.11 void vVectorTransformV3 (VECTOR3 * *vd*, VECTOR3 * *vs*, int32_t *count*)

Transform vector array with current matrix.

This function transforms vectors in vector array vs with the current matrix and stores the resulting vectors in vector array vd.

Parameters:

vd Pointer to destination vector array.

vs Pointer to source 3D vector array.

count Number of vectors to transform.

5.32.1.12 void vVectorTransformV4 (VECTOR4 * *vd*, VECTOR3 * *vs*, int32_t *count*)

Transform vector array with current matrix.

This function transforms vectors in vector array vs with the current matrix and the projection matrix and stores the resulting vectors in the 4D vector array vd.

Parameters:

vd Pointer to destination 4D vector array.

vs Pointer to source 3D vector array.

count Number of vectors to transform.

5.33 Capabilities

The mophun API allows a game to query the system for certain features, for example checking the screen size.

Modules

- System capabilities
- Video capabilities
- Input capabilities
- Sound capabilities
- Communication capabilities

Data Structures

- union CAPS

  Union of all capability structures.

Defines

- #define **SIZEOF_SOUNDCAPS** 6
- #define **SIZEOF_NEW_SOUNDCAPS** 12
- #define **MASK_SOUNDCAPS** 1

Enumerations

- enum CapsQuery {

  CAPS_VIDEO, CAPS_INPUT, CAPS_SOUND, CAPS_COMM,

  CAPS_SYSTEM, CAPS_VENDOR = 100 }

  Capability query type.

Functions

- int32_t vGetCaps (enum CapsQuery query, void *buf)

  Query capabilities.

5.33.1 Detailed Description

The mophun API allows a game to query the system for certain features, for example checking the screen size.

5.33.2 Enumeration Type Documentation

5.33.2.1 enum CapsQuery

Capability query type.

Enumeration values:

CAPS_VIDEO Query video capabilities.
See Video capabilities.

CAPS_INPUT Query input capabilities.
See Input capabilities.

CAPS_SOUND Query sound capabilities.
See Sound capabilities.

CAPS_COMM Query communication capabilities.
See Communication capabilities.

CAPS_SYSTEM Query system capabilities.
See System capabilities.

CAPS_VENDOR Vendor specific capabilities.

5.33.3 Function Documentation

5.33.3.1 int32_t vGetCaps (enum CapsQuery *query*, void * *buf*)

Query capabilities.

This function performs a query about the specified group of capabilities.

Parameters:

query The type of query to perform.

buf Pointer to capability structure.

Returns:

Non-zero on success, 0 on failure.

See also:

SYSCAPS, VIDEOCAPS, SOUNDCAPS, COMMCAPS, INPUTCAPS.

5.34 System capabilities

Data Structures

- struct SYSCAPS

  System capabilities.

System capability flags.

- #define SYSTEM_UNICODE 0x0001

  The system supports unicode.

- #define SYSTEM_VIBRATE 0x0004

  The system has a vibrator.

- #define SYSTEM_BIGENDIAN 0x0010

  The system is big endian.

Vendor identifiers

- #define VENDOR_UNKNOWN 0

  Unknown vendor.

- #define VENDOR_SYNERGENIX 1

  Synergenix Interactive AB.

- #define VENDOR_PALM 2

  Palm, Inc.

- #define VENDOR_SONYERICSSON 3

  Sony Ericsson Mobile Communications.

- #define VENDOR_NOKIA 4

  Nokia.

- #define VENDOR_MOTOROLA 5

  Motorola, Inc.

Model numbers

- #define UNKNOWN_UNKNOWN 0

  Unknown model.

- #define UNKNOWN_UNIX 1

  Unknown unix environment.

- #define UNKNOWN_WINDOWS 2
  Unknown windows environment.
- #define UNKNOWN_POCKETPC 3
  Unknown PocketPC environment.
- #define SONYERICSSON_T300 0
  SonyEricsson T300.
- #define SONYERICSSON_T610 2
  SonyEricsson T610.
- #define NOKIA_7650 3
  Nokia Series60 7650.
- #define SONYERICSSON_P800 4
  SonyEricsson P800.
- #define SONYERICSSON_T226 5
  SonyEricsson T226.
- #define MOTOROLA_A920 6
  Motorola A920.
- #define NOKIA_3650 7
  Nokia Series60 3650.
- #define NOKIA_NGAGE 8
  Nokia Series60 N-Gage.

Defines

- #define MAKE_DEVID(vendor, num) ((vendor) | ((uint32_t)(num) << 16))
  Create devide id.
- #define DEVICE_VENDOR(id) ((uint32_t)(id) & 0xffff)
  Get device vendor.
- #define DEVICE_NUMBER(id) ((uint32_t)(id) >> 16)
  Get device number.

5.34.1 Define Documentation

5.34.1.1 #define DEVICE_NUMBER(id) ((uint32_t)(id) >> 16)

Get device number.

This macro extracts the device number (model) from a full 32-bit device id. See Models for a list of model numbers.

Parameters:
id Full 32-bit device id.

Returns:
Device (model) number.

See also:
MAKE_DEVID, DEVICE_VENDOR.

5.34.1.2 #define DEVICE_VENDOR(id) ((uint32_t)(id) & 0xffff)

Get device vendor.

This macro extracts the vendor identifier from a full 32-bit device id. See Vendors for a list of vendor identifiers.

Parameters:
id Full 32-bit device id.

Returns:
Vendor id.

See also:
MAKE_DEVID, DEVICE_NUMBER.

5.34.1.3 #define MAKE_DEVID(vendor, num) ((vendor) | ((uint32_t)(num) << 16))

Create devide id.

This macro creates a 32-bit device identifier from a vendor identifier and a model number.

Parameters:
vendor Vendor id, see Vendors.
num Model number, see Models.

Returns:
Full 32-bit device identifier.

See also:
DEVICE_VENDOR, DEVICE_NUMBER.

5.34.1.4 #define MOTOROLA_A920 6

Motorola A920.

5.34.1.5 #define NOKIA_3650 7

Nokia Series60 3650.

5.34.1.6 #define NOKIA_7650 3

Nokia Series60 7650.

5.34.1.7 #define NOKIA_NGAGE 8

Nokia Series60 N-Gage.

5.34.1.8 #define SONYERICSSON_P800 4

SonyEricsson P800.

5.34.1.9 #define SONYERICSSON_T226 5

SonyEricsson T226.

5.34.1.10 #define SONYERICSSON_T300 0

SonyEricsson T300.

5.34.1.11 #define SONYERICSSON_T610 2

SonyEricsson T610.

5.34.1.12 #define SYSTEM_BIGENDIAN 0x0010

The system is big endian.

5.34.1.13 #define SYSTEM_UNICODE 0x0001

The system supports unicode.

5.34.1.14 #define SYSTEM_VIBRATE 0x0004

The system has a vibrator.

5.34.1.15 #define UNKNOWN_POCKETPC 3

Unknown PocketPC environment.

5.34.1.16 #define UNKNOWN_UNIX 1

Unknown unix environment.

5.34.1.17 #define UNKNOWN_UNKNOWN 0

Unknown model.

5.34.1.18 #define UNKNOWN_WINDOWS 2

Unknown windows environment.

5.34.1.19 #define VENDOR_MOTOROLA 5

Motorola, Inc.

5.34.1.20 #define VENDOR_NOKIA 4

Nokia.

5.34.1.21 #define VENDOR_PALM 2

Palm, Inc.

5.34.1.22 #define VENDOR_SONYERICSSON 3

Sony Ericsson Mobile Communications.

5.34.1.23 #define VENDOR_SYNERGENIX 1

Synergenix Interactive AB.

5.34.1.24 #define VENDOR_UNKNOWN 0

Unknown vendor.

5.35 Video capabilities

Data Structures

- struct VIDEOCAPS

  Video capabilities.

Defines

- #define VCAPS_3D 0x1000U

  The system supports mophun 3D.

- #define VCAPS_ORIENTATION 0x2000U

  The system supports changing orientation.

5.35.1 Define Documentation

5.35.1.1 #define VCAPS_3D 0x1000U

The system supports mophun 3D.

see ...

5.35.1.2 #define VCAPS_ORIENTATION 0x2000U

The system supports changing orientation.

See vSetOrientation for information on how to change orientation.

5.36 Input capabilities

Data Structures

- struct INPUTCAPS

  Input capabilities.

Input flags

- #define ICAPS_POINTER 0x0001

  The device has a mouse or touchscreen.

- #define ICAPS_JOYSTICK 0x0002

  The device has an analog joystick.

- #define ICAPS_ASCII 0x0004

  The device supports ASCII character input.

- #define ICAPS_NUMERIC_KEYPAD 0x0008

  The device has a numeric keypad.

- #define ICAPS_ONSCREEN 0x0010

  The device has onscreen input controls, currently only SONYERICSSON_P800.

5.36.1 Define Documentation

5.36.1.1 #define ICAPS_ASCII 0x0004

The device supports ASCII character input.

5.36.1.2 #define ICAPS_JOYSTICK 0x0002

The device has an analog joystick.

5.36.1.3 #define ICAPS_NUMERIC_KEYPAD 0x0008

The device has a numeric keypad.

5.36.1.4 #define ICAPS_ONSCREEN 0x0010

The device has onscreen input controls, currently only SONYERICSSON_P800.

5.36.1.5 #define ICAPS_POINTER 0x0001

The device has a mouse or touchscreen.

5.37 Sound capabilities

Data Structures

- struct SOUNDCAPS

  Sound capabilities.

- struct SOUNDCONFIG

  PCM output settings.

Sound flags

- #define SCAPS_BEEP 0x0001

  The device has a beeper (tone generator).

- #define SCAPS_WAVE 0x0002

  The device has wave output (PCM) support.

- #define SCAPS_STEREO 0x0004

  The device supports stereo sound output.

- #define SCAPS_MIDI 0x0008

  The device has support for midi playback.

- #define SCAPS_MONO 0x0010

  The device supports mono sound output.

- #define SCAPS_8BIT 0x0020

  The device supports 8-bit PCM sound output.

- #define SCAPS_16BIT 0x0040

  The device supports 16-bit PCM sound output.

- #define SCAPS_CTRLFREQUENCY 0x0080

  Device supports changing PCM frequency settings.

- #define SCAPS_CTRLPAN 0x0100

  The device supports panning control.

- #define SCAPS_CTRLVOLUME 0x0200

  Platform supports changing PCM volume settings.

- #define SCAPS_CTRLMASTERVOLUME 0x0400

  Platform supports changing PCM master volume settings.

5.37.1 Define Documentation

5.37.1.1 #define SCAPS_16BIT 0x0040

The device supports 16-bit PCM sound output.

Since:
API version 1.50

5.37.1.2 #define SCAPS_8BIT 0x0020

The device supports 8-bit PCM sound output.

Since:
API version 1.50

5.37.1.3 #define SCAPS_BEEP 0x0001

The device has a beeper (tone generator).

5.37.1.4 #define SCAPS_CTRLFREQUENCY 0x0080

Device supports changing PCM frequency settings.

Since:
API version 1.50

5.37.1.5 #define SCAPS_CTRLMASTERVOLUME 0x0400

Platform supports changing PCM master volume settings.

Since:
API version 1.50

5.37.1.6 #define SCAPS_CTRLPAN 0x0100

The device supports panning control.

Since:
API version 1.50

5.37.1.7 #define SCAPS_CTRLVOLUME 0x0200

Platform supports changing PCM volume settings.

Since:
API version 1.50

5.37.1.8 #define SCAPS_MIDI 0x0008

The device has support for midi playback.

5.37.1.9 #define SCAPS_MONO 0x0010

The device supports mono sound output.

Since:
API version 1.50

5.37.1.10 #define SCAPS_STEREO 0x0004

The device supports stereo sound output.

5.37.1.11 #define SCAPS_WAVE 0x0002

The device has wave output (PCM) support.

5.38 Communication capabilities

Data Structures

- struct COMMCAPS

  Communication capabilities.

Communication flags

- #define CCAPS_FILE 0x0001

  System supports file-system access.

- #define CCAPS_TCP 0x0002

  System supports TCP/IP network streams.

- #define CCAPS_UDP 0x0004

  System supports UDP datagram streams.

- #define CCAPS_BLUETOOTH 0x0008

  System supports bluetooth RFCOMM streams.

- #define CCAPS_BT CCAPS_BLUETOOTH

  Alias for CCAPS_BLUETOOTH.

- #define CCAPS_IR 0x0010

  System supports IrDA streams.

- #define CCAPS_SMS 0x0020

  System supports sending and receiving SMS's.

- #define CCAPS_CABLE 0x0040

  System supports serial cable streams.

- #define CCAPS_HTTP 0x0080

  System supports HTTP streams.

- #define CCAPS_OBEX 0x8000

  System uses the OBEX protocol for BT, IR and CABLE.

5.38.1 Define Documentation

5.38.1.1 #define CCAPS_BLUETOOTH 0x0008

System supports bluetooth RFCOMM streams.

5.38.1.2 #define CCAPS_BT CCAPS_BLUETOOTH

Alias for CCAPS_BLUETOOTH.

5.38.1.3 #define CCAPS_CABLE 0x0040

System supports serial cable streams.

5.38.1.4 #define CCAPS_FILE 0x0001

System supports file-system access.

5.38.1.5 #define CCAPS_HTTP 0x0080

System supports HTTP streams.

5.38.1.6 #define CCAPS_IR 0x0010

System supports IrDA streams.

5.38.1.7 #define CCAPS_OBEX 0x8000

System uses the OBEX protocol for BT, IR and CABLE.

5.38.1.8 #define CCAPS_SMS 0x0020

System supports sending and receiving SMS's.

5.38.1.9 #define CCAPS_TCP 0x0002

System supports TCP/IP network streams.

5.38.1.10 #define CCAPS_UDP 0x0004

System supports UDP datagram streams.

5.39 Utility macros

Defines

- #define WaitKey(key) while ((vGetButtonData() & key) == 0)

  Wait until a key is pressed.

- #define WaitNotKey(key) while (vGetButtonData() & key)

  Wait until a key is released.

- #define WaitTicks(ms)

  Wait for a specified number of milli-seconds.

- #define msSleep WaitTicks

  Wait for a specified number of milli-seconds.

- #define abort() vTerminateVMGP()

  Abort execution.

- #define exit(code) vTerminateVMGP()

  Exit.

- #define BeepOff() vBeep(0,0)

  Turn off beeper.

5.39.1 Define Documentation

5.39.1.1 #define abort() vTerminateVMGP()

Abort execution.

5.39.1.2 #define BeepOff() vBeep(0,0)

Turn off beeper.

5.39.1.3 #define exit(code) vTerminateVMGP()

Exit.

5.39.1.4 #define msSleep WaitTicks

Wait for a specified number of milli-seconds.

5.39.1.5 #define WaitKey(key) while ((vGetButtonData() & key) == 0)

Wait until a key is pressed.

Parameters:
key The key to wait for, see Key codes.

5.39.1.6 #define WaitNotKey(key) while (vGetButtonData() & key)

Wait until a key is released.

Parameters:
key The key to wait for, see Key codes.

5.39.1.7 #define WaitTicks(ms)

Value:

```
{ \
        unsigned int _t0 = vGetTickCount(); \
        unsigned int _d = (ms); \
        while ((vGetTickCount() - _t0) < _d); \
}
```

Wait for a specified number of milli-seconds.

Parameters:
ms The number of milli-seconds to wait.

5.40 Graphics formats

Mophun supports a large number of graphics formats.

Defines

- #define **VCAPS_FORMAT_MASK** 0xFF
- #define VCAPS_GRY2 0

  Monochrome.

- #define VCAPS_GRY4 1

  4-level grayscale

- #define VCAPS_GRY16 2

  16-level grayscale

- #define VCAPS_IND2 3

  2-color indexed

- #define VCAPS_IND4 4

  4-color indexed

- #define VCAPS_IND16 5

  16-color indexed

- #define VCAPS_IND256 6

  256-color indexed

- #define VCAPS_RGB332 7

  RGB332 direct color.

- #define VCAPS_16BPP 8

  RGB565 direct color.

- #define VCAPS_15BPP 9

  RGB555 direct color.

- #define VCAPS_24BPP 10

  RGB888 direct color.

- #define VCAPS_RGBA 11

  RGB888A direct color.

- #define VCAPS_4K 12

  RGB444 direct color.

5.40.1 Detailed Description

Mophun supports a large number of graphics formats.

This is a list of possible graphics formats.

5.40.2 Define Documentation

5.40.2.1 #define VCAPS_15BPP 9

RGB555 direct color.

5.40.2.2 #define VCAPS_16BPP 8

RGB565 direct color.

5.40.2.3 #define VCAPS_24BPP 10

RGB888 direct color.

5.40.2.4 #define VCAPS_4K 12

RGB444 direct color.

5.40.2.5 #define VCAPS_GRY16 2

16-level grayscale

5.40.2.6 #define VCAPS_GRY2 0

Monochrome.

5.40.2.7 #define VCAPS_GRY4 1

4-level grayscale

5.40.2.8 #define VCAPS_IND16 5

16-color indexed

5.40.2.9 #define VCAPS_IND2 3

2-color indexed

5.40.2.10 #define VCAPS_IND256 6

256-color indexed

5.40.2.11 #define VCAPS_IND4 4

4-color indexed

5.40.2.12 #define VCAPS_RGB332 7

RGB332 direct color.

5.40.2.13 #define VCAPS_RGBA 11

RGB888A direct color.

5.41 Sound API

Since version 1.50 mophun supports playback of PCM/ADPCM sounds on multiple channels.

Return codes

- #define SND_OK 0

  Success.

- #define SND_ERR -1

  Failure.

Limits

- #define SNDVOLUME_MIN 0

  No sound.

- #define SNDVOLUME_MAX 127

  Maximum volume.

- #define SNDPAN_LEFT 0

  Left-most pan position.

- #define SNDPAN_RIGHT 127

  Right-most pan position.

- #define SNDPAN_CENTER 64

  Center pan position.

- #define SNDFREQUENCY_MIN 100

  Lowest sound frequency allowed.

- #define SNDFREQUENCY_MAX 100000

  Highest sound frequency allowed.

- #define SNDPRIORITY_MIN 0

  Lowest sound priority.

- #define SNDPRIORITY_MAX 127

  Highest sound priority.

- #define SNDPRIORITY_DEFAULT 64

  Default sound priority.

Sound status

- #define SNDSTATUS_PLAYING 0x01

  Sound is playing.

- #define SNDSTATUS_PAUSED 0x02

  Sound is paused.

- #define SNDSTATUS_LOOPING 0x04

  Sound is looping.

- #define SNDSTATUS_CASHED 0x08

  Internal.

- #define SNDSTATUS_LOADED 0x10

  Internal.

Playback parameters

- #define SNDPLAY_LOOPING ((uint16_t)((int32_t)-1))

  Loop the sound continously.

- #define SNDPLAY_OVERRIDE (1 << 16)

  Overrides the currently playing sound.

Sound load flags

- #define SNDLOAD_STREAM 0

  Load sound from a stream.

- #define SNDLOAD_RESOURCE 1

  Load sound from a resource.

- #define SNDLOAD_FILE 2

  Load sound from a file.

Sound formats.

- #define **SNDFORMAT_PCM** 0x01 /** PCM sound format. */
- #define **SNDFORMAT_ADPCM** 0x02 /** ADPCM sound format. */

Sound control macros

- #define vSoundPlay(hSound, flags) vSoundCtrlEx(hSound, SNDCTRL_PLAY, flags)

  Play sound.

- #define vSoundStop(hSound) vSoundCtrl(hSound, SNDCTRL_STOP)

  Stop sound.

- #define vSoundPause(hSound) vSoundCtrl(hSound, SNDCTRL_PAUSE)

  Pause sound.

- #define vSoundResume(hSound) vSoundCtrl(hSound, SNDCTRL_RESUME)

  Resume sound.

- #define vSoundSetVolume(hSound, volume) vSoundCtrlEx(hSound, SNDCTRL_VOLUME, volume)

  Set sound volume.

- #define vSoundSetPan(hSound, pan) vSoundCtrlEx(hSound, SNDCTRL_PAN, pan)

  Set sound panning.

- #define vSoundSetFrequency(hSound, freq) vSoundCtrlEx(hSound, SNDCTRL_FREQ, freq)

  Set sound playback frequency.

- #define vSoundSetPosition(hSound, position) vSoundCtrlEx(hSound, SNDCTRL_POSITION, position)

  Set sound playback position.

- #define vSoundSetPriority(hSound, priority) vSoundCtrlEx(hSound, SNDCTRL_PRIORITY, priority)

  Set sound priority.

- #define vSoundSetMasterVolume(volume) vSoundCtrlEx(0, SNDCTRL_MASTERVOLUME, volume)

  Set master volume.

- #define vSoundGetStatus(hSound) vSoundCtrl(hSound, SNDCTRL_STATUS)

  Get sound status.

- #define vSoundStopLooping(hSound) vSoundCtrl(hSound, SNDCTRL_STOPLOOPING)

  Stop sound looping.

- #define vSoundSetParameters(hSound, volume, pan, frequency) vSoundCtrlEx(hSound, SNDCTRL_PARAMETERS, ((volume << 24) | (pan << 16) | (frequency)))

  Set sound parameters.

- #define vSoundLoadFile(filename) vSoundLoad((int32_t)filename, SNDLOAD_FILE)

  Load sound from file.

- #define vSoundLoadResource(idx) vSoundLoad(idx, SNDLOAD_RESOURCE)

Load sound from resource.

- #define vSoundLoadStream(handle) vSoundLoad(handle, SNDLOAD_STREAM)
  Load sound from a stream.

Typedefs

- typedef uint32_t HSOUND
  Sound handle.

Enumerations

- enum SNDCTRL {
  SNDCTRL_NULL, **SNDCTRL_PLAY**, SNDCTRL_STOP, SNDCTRL_PAUSE,
  SNDCTRL_RESUME, SNDCTRL_VOLUME, SNDCTRL_FREQ, SNDCTRL_PAN,
  SNDCTRL_STATUS, SNDCTRL_STOPLOOPING, SNDCTRL_POSITION, SNDCTRL_-PARAMETERS,
  SNDCTRL_MASTERVOLUME, SNDCTRL_PRIORITY }
  Sound control codes.

Functions

- int32_t vSoundInit (void)
  Initialize sound system.
- int32_t vSoundDispose (void)
  Close sound system.
- int32_t vSoundUpload (HSOUND hSound)
  Upload sound to hardware buffer.
- HSOUND vSoundGetHandle (const void *soundData)
  Create sound handle from sound data.
- HSOUND vSoundLoad (int32_t handle, int32_t flags)
  Load sound and create sound handle.
- int32_t vSoundDisposeHandle (HSOUND hSound)
  Release sound handle.
- int32_t vSoundCtrl (HSOUND hSound, int32_t msg)
  Sound control message.
- int32_t vSoundCtrlEx (HSOUND hSound, int32_t msg, int32_t parameters)
  Extended sound control message.

5.41.1 Detailed Description

Since version 1.50 mophun supports playback of PCM/ADPCM sounds on multiple channels.

The sound API is initialized by calling vSoundInit(), after successful initialization sounds may be loaded by calling one of the sound loader functions, for example vSoundGetHandle(). To start playback of a sound vSoundPlay() is called. When the sound API is no longer needed, the user should call vSoundDispose().

Since:
API version 1.50

5.41.2 Define Documentation

5.41.2.1 #define SND_ERR -1

Failure.

5.41.2.2 #define SND_OK 0

Success.

5.41.2.3 #define SNDFREQUENCY_MAX 100000

Highest sound frequency allowed.

5.41.2.4 #define SNDFREQUENCY_MIN 100

Lowest sound frequency allowed.

5.41.2.5 #define SNDLOAD_FILE 2

Load sound from a file.

The handle parameter passed to vSoundLoad() is interpreted as a filename from which the sound will be loaded.

5.41.2.6 #define SNDLOAD_RESOURCE 1

Load sound from a resource.

The handle parameter passed to vSoundLoad() is interpreted as a resource index into the games internal resources from which the sound will be loaded.

5.41.2.7 #define SNDLOAD_STREAM 0

Load sound from a stream.

The handle parameter passed to vSoundLoad() is interpreted as a stream handle from which the sound will be loaded. Only file and resource streams are allowed.

5.41.2.8 #define SNDPAN_CENTER 64

Center pan position.

5.41.2.9 #define SNDPAN_LEFT 0

Left-most pan position.

5.41.2.10 #define SNDPAN_RIGHT 127

Right-most pan position.

5.41.2.11 #define SNDPLAY_LOOPING ((uint16_t)((int32_t)-1))

Loop the sound continously.

5.41.2.12 #define SNDPLAY_OVERRIDE (1 << 16)

Overrides the currently playing sound.
If there are no free channels, the sound with lowest priority is cancelled.

5.41.2.13 #define SNDPRIORITY_DEFAULT 64

Default sound priority.

5.41.2.14 #define SNDPRIORITY_MAX 127

Highest sound priority.

5.41.2.15 #define SNDPRIORITY_MIN 0

Lowest sound priority.

5.41.2.16 #define SNDSTATUS_CASHED 0x08

Internal.

5.41.2.17 #define SNDSTATUS_LOADED 0x10

Internal.

5.41.2.18 #define SNDSTATUS_LOOPING 0x04

Sound is looping.

5.41.2.19 #define SNDSTATUS_PAUSED 0x02

Sound is paused.

5.41.2.20 #define SNDSTATUS_PLAYING 0x01

Sound is playing.

5.41.2.21 #define SNDVOLUME_MAX 127

Maximum volume.

5.41.2.22 #define SNDVOLUME_MIN 0

No sound.

5.41.2.23 #define vSoundGetStatus(hSound) vSoundCtrl(hSound, SNDCTRL_STATUS)

Get sound status.

Parameters:
hSound Handle to a sound.

Returns:
The sounds current status flags, SND_ERR on failure.

See also:
SNDCTRL_STATUS.

5.41.2.24 #define vSoundLoadFile(filename) vSoundLoad((int32_t)filename, SNDLOAD_FILE)

Load sound from file.

Parameters:
filename The name of the file from which the sound will be loaded.

Returns:
A sound handle, SND_ERR on failure.

See also:
vSoundLoadResource, vSoundLoadStream, vSoundLoad, vSoundGetHandle.

5.41.2.25 #define vSoundLoadResource(idx) vSoundLoad(idx, SNDLOAD_RESOURCE)

Load sound from resource.

This function loads a sound from a resource in the game.

Parameters:
idx The resource index from which the sound will be loaded.

Returns:
A sound handle, SND_ERR on failure.

See also:
vSoundLoadFile, vSoundLoadStream, vSoundLoad, vSoundGetHandle.

5.41.2.26 #define vSoundLoadStream(handle) vSoundLoad(handle, SNDLOAD_STREAM)

Load sound from a stream.

This function loads a sound from a stream. The stream must be either a resource or a file.

Multiple calls to this function on the same stream may be used to load several consecutive sounds. The stream handle must be closed by the caller.

Parameters:
handle The stream from which the sound will be loaded.

Returns:
A sound handle, SND_ERR on failure.

See also:
vSoundLoadFile, vSoundLoadResource, vSoundLoad, vSoundGetHandle.

5.41.2.27 #define vSoundPause(hSound) vSoundCtrl(hSound, SNDCTRL_PAUSE)

Pause sound.

Parameters:
hSound Handle to a sound.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_PAUSE.

5.41.2.28 #define vSoundPlay(hSound, flags) vSoundCtrlEx(hSound, SNDCTRL_PLAY, flags)

Play sound.

Parameters:
hSound Handle to a sound.
flags Playback flags, this includes priority override and loopcount. See SoundPlayParms.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_PLAY.

5.41.2.29 #define vSoundResume(hSound) vSoundCtrl(hSound, SNDCTRL_RESUME)

Resume sound.

Parameters:
hSound Handle to a sound.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_RESUME.

5.41.2.30 #define vSoundSetFrequency(hSound, freq) vSoundCtrlEx(hSound, SNDCTRL_FREQ, freq)

Set sound playback frequency.

Parameters:
hSound Handle to a sound.

freq The new playback frequency in the range SNDFREQUENCY_MIN to SNDFREQUENCY_-MAX.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_FREQ.

5.41.2.31 #define vSoundSetMasterVolume(volume) vSoundCtrlEx(0, SNDCTRL_MASTERVOLUME, volume)

Set master volume.

Parameters:
volume A volume in the range SNDVOLUME_MIN to SNDVOLUME_MAX.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_MASTERVOLUME.

5.41.2.32 #define vSoundSetPan(hSound, pan) vSoundCtrlEx(hSound, SNDCTRL_PAN, pan)

Set sound panning.

Parameters:
hSound Handle to a sound.

pan A panning value in the range SNDPAN_LEFT to SNDPAN_RIGHT, SNDPAN_CENTER is neutral position.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_PAN.

5.41.2.33 #define vSoundSetParameters(hSound, volume, pan, frequency) vSoundCtrlEx(hSound, SNDCTRL_PARAMETERS, ((volume << 24) | (pan << 16) | (frequency)))

Set sound parameters.

Parameters:
hSound Handle to a sound.

volume New sound volume in the range SNDVOLUME_MIN to SNDVOLUME_MAX.

pan New panning value in the range SNDPAN_LEFT to SNDPAN_RIGHT, SNDPAN_CENTER is neutral position.

frequency New playback frequency in the range SNDFREQUENCY_MIN to SNDFREQUENCY_MAX.

Returns:
SND_OK on success, SND_ERR on failure.
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_STOPLOOPING.

5.41.2.34 #define vSoundSetPosition(hSound, position) vSoundCtrlEx(hSound, SNDCTRL_POSITION, position)

Set sound playback position.

Parameters:
hSound Handle to a sound.

position The new playback position, expressed in sample frames.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_POSITION.

5.41.2.35 #define vSoundSetPriority(hSound, priority) vSoundCtrlEx(hSound, SNDCTRL_PRIORITY, priority)

Set sound priority.

Parameters:

hSound Handle to a sound.

priority The new sound priority in the range SNDPRIORITY_MIN to SNDPRIORITY_MAX, default is SNDPRIORITY_DEFAULT.

Returns:

SND_OK on success, SND_ERR on failure.

See also:

SNDCTRL_PRIORITY.

5.41.2.36 #define vSoundSetVolume(hSound, volume) vSoundCtrlEx(hSound, SNDCTRL_VOLUME, volume)

Set sound volume.

Parameters:

hSound Handle to a sound.

volume A volume in the range SNDVOLUME_MIN to SNDVOLUME_MAX.

Returns:

SND_OK on success, SND_ERR on failure.

See also:

SNDCTRL_VOLUME.

5.41.2.37 #define vSoundStop(hSound) vSoundCtrl(hSound, SNDCTRL_STOP)

Stop sound.

Parameters:

hSound Handle to a sound.

Returns:

SND_OK on success, SND_ERR on failure.

See also:

SNDCTRL_STOP.

5.41.2.38 #define vSoundStopLooping(hSound) vSoundCtrl(hSound, SNDCTRL_STOPLOOPING)

Stop sound looping.

Parameters:
hSound Handle to a sound.

Returns:
SND_OK on success, SND_ERR on failure.

See also:
SNDCTRL_STOPLOOPING.

5.41.3 Typedef Documentation

5.41.3.1 typedef uint32_t HSOUND

Sound handle.

This type is used to refer to a sound.

5.41.4 Enumeration Type Documentation

5.41.4.1 enum SNDCTRL

Sound control codes.

A sound control code is used to control various aspects of sound playback. Sound control codes are handled by the functions vSoundCtrl() and vSoundCtrlEx(). vSoundCtrl takes a sound handle and a sound control code, vSoundCtrlEx() takes an additional parameters argument which contains information needed to perform the specified action. If the parameters argument to vSoundCtrlEx() is -1 the current parameters are returned.

See also:
vSoundCtrl, vSoundCtrlEx.

Enumeration values:

SNDCTRL_NULL Do nothing.

Returns SND_OK if sound system is initialized, otherwise SND_ERR.

SNDCTRL_STOP Stop playback of sound.

See also:
vSoundStop, vSoundCtrl.

SNDCTRL_PAUSE Pause playback of sound.

See also:
vSoundPause, vSoundCtrl.

SNDCTRL_RESUME Resume playback of sound.

See also:
vSoundResume, vSoundCtrl.

SNDCTRL_VOLUME Set volume on channel.

Parameters is a value between SNDVOLUME_MIN and SNDVOLUME_MAX.

See also:

vSoundSetVolume, vSoundCtrlEx.

SNDCTRL_FREQ Set playback frequency of sound.

Parameters is a frequency between SNDFREQUENCY_MIN and SNDFREQUENCY_MAX.

See also:

vSoundSetFrequency, vSoundCtrlEx.

SNDCTRL_PAN Set panning of sound.

Only supported with stereo output. Parameters is a pan value between SNDPAN_LEFT and SNDPAN_RIGHT, SNDPAN_CENTER indicates no panning.

See also:

vSoundSetPan, vSoundCtrlEx.

SNDCTRL_STATUS Get current status of sound.

See SoundStatus for a list of possible values.

See also:

vSoundGetStatus, vSoundCtrl.

SNDCTRL_STOPLOOPING Stop looping a sound.

The sound continues playing until the end of the sound is reached.

See also:

vSoundStopLooping, vSoundCtrl.

SNDCTRL_POSITION Set sound position.

Parameteres is a position within the sound in sample frames.

See also:

vSoundSetPosition, vSoundCtrlEx.

SNDCTRL_PARAMETERS Set volume, pan and frequency of sound.

Parameters is encoded as follows: `(volume << 24) | (Pan << 16) | frequency)`.

See also:

vSoundSetParameters, vSoundCtrlEx.

SNDCTRL_MASTERVOLUME Set/get master volume.

Parameters is value between SNDVOLUME_MIN and SNDVOLUME_MAX. The hSound argument is ignored.

See also:

vSoundSetMasterVolume, vSoundCtrlEx.

SNDCTRL_PRIORITY Set sound priority.

Parameters is a priority value between SNDPRIORITY_MIN and SNDPRIORITY_MAX.

See also:

vSoundSetPriority, vSoundCtrlEx.

5.41.5 Function Documentation

5.41.5.1 int32_t vSoundCtrl (HSOUND *hSound*, int32_t *msg*)

Sound control message.

This function sends a control message to the sound system. The preferred way to send control messages is by using pre-defined macros, see SoundMacros.

Parameters:

hSound A sound handle.

msg A control message code, see SNDCTRL.

Returns:

SND_OK on success, SND_ERR on failure.

5.41.5.2 int32_t vSoundCtrlEx (HSOUND *hSound*, int32_t *msg*, int32_t *parameters*)

Extended sound control message.

This function sends a control message and parameters to the sound system. The preferred way to send control messages is by using pre-defined macros, see SoundMacros.

Parameters:

hSound A sound handle.

msg A control message code, see SNDCTRL.

parameters The control parameters, -1 is used to return the current parameters.

Returns:

SND_OK on success, SND_ERR on failure. If parameters was -1, the current parameters are typically returned.

5.41.5.3 int32_t vSoundDispose (void)

Close sound system.

This function should be called when the sound API is no longer needed. All sound handles are automatically released and are no longer valid when the function returns.

Returns:

SND_OK on success, SND_ERR on failure.

See also:

vSoundInit.

5.41.5.4 int32_t vSoundDisposeHandle (HSOUND *hSound*)

Release sound handle.

Releases a sound handle obtained with vSoundGetHandle() or vSoundLoad(). Also releases it from hardware if it is loaded with vSoundUpload into a dedicated hardware sound buffer.

Parameters:

hSound A sound handle.

Returns:

SND_OK on success, SND_ERR on failure.

5.41.5.5 HSOUND vSoundGetHandle (const void * *soundData*)

Create sound handle from sound data.

This function creates a sound handle for sound data stored in memory. The handle returned is used for controlling playback and for setting various parameters on the sound. The sound handle must be released by calling vSoundDisposeHandle().

Parameters:
soundData Pointer to a sound data header.

Returns:
A sound handle, SND_ERR on failure.

See also:
vSoundLoad, vSoundDisposeHandle.

5.41.5.6 int32_t vSoundInit (void)

Initialize sound system.

This function initializes the sound system. It must be called before any other sound API functions are called.

Returns:
SND_OK if initialization was successful, SND_ERR on failure.

See also:
vSoundDispose, vSoundGetHandle, vSoundLoad.

5.41.5.7 HSOUND vSoundLoad (int32_t *handle*, int32_t *flags*)

Load sound and create sound handle.

This function can be used to load a sound from a stream, resource or file. The preferred way of using this function is with the macros vSoundLoadFile(), vSoundLoadResource() and vSoundLoadStream().

Parameters:
handle A stream handle, a resource index or a filename cast to a int32_t.

flags How to interpret the handle parameter, see SoundLoadFlags.

Returns:
A sound handle, SND_ERR on failure.

See also:
vSoundLoadFile, vSoundLoadResource, vSoundLoadStream, vSoundGetHandle, vSoundDisposeHandle.

Remarks:
When loading sounds from a stream, multiple calls to this function may be used to load several consecutive sounds. The stream handle must be closed by the caller.

5.41.5.8 int32_t vSoundUpload (HSOUND *hSound*)

Upload sound to hardware buffer.

Used to upload a sound into a dedicated hardware sound buffer if the platform has support for this.

Parameters:

hSound A sound handle.

Returns:

SND_OK on success, SND_ERR on failure.

See also:

vSoundGetHandle, vSoundLoad.

5.42 Stream I/O

Mophun streams are ways of transferring data, for across networks, wireless links or files.

Modules

- HTTP streams

  Mophun supports streams that connect to an HTTP server.

- Stream types
- Stream Modes
- Seek Modes
- File helper macros

  Mophun defines a number of file macros that may be used instead of the low-level stream functions.

- TCP macros

  Mophun defines a number of macros that may be used for establishing TCP connections instead of the low-level stream functions.

- Resource macros

  Mophun defines a number of macros that may be used to handle resources instead of using the low-level stream functions.

- SMS streams

  SMS streams send and recieve SMS messages.

Data Structures

- struct VDGRAM

  UDP datagram address.

Defines

- #define **STREAM_TYPE_MASK** 0xFF
- #define vStreamWriteTo vStreamTo

  Alias for vStreamTo().

- #define vStreamReadFrom vStreamFrom

  Alias for vStreamFrom().

- #define ntohl(netlong) htonl(netlong)

  Convert from network-endian (big-endian) to host-endian.

- #define ntohs(netshort) htons(netshort)

  Convert from network-endian (big-endian) to host-endian.

- #define **STREAM_MAX_RESOURCES** 0x10000

Functions

- int32_t vStreamOpen (const char *name, int32_t mode)

  Create a stream.

- void vStreamClose (int32_t handle)

  Close a stream.

- int32_t vStreamRead (int32_t handle, void *buf, int32_t count)

  Read data from a stream.

- int32_t vStreamWrite (int32_t handle, const void *buf, int32_t count)

  Write data to a stream.

- int32_t vStreamSeek (int32_t handle, int32_t where, int32_t whence)

  Seek in stream.

- int32_t vStreamMode (int32_t handle)

  Get stream type and mode.

- int32_t vStreamReady (int32_t handle, int32_t mode)

  Check stream state.

- int32_t vStreamFrom (int32_t handle, void *buf, int32_t count, void *args)

  Read from stream with attributes.

- int32_t vStreamTo (int32_t handle, void *buf, int32_t count, void *args)

  Write to stream with attributes.

- unsigned long htonl (unsigned long hostlong)

  Convert from host-endian to network-endian (big-endian).

- unsigned short htons (unsigned short hostshort)

  Convert from host-endian to network-endian (big-endian).

5.42.1 Detailed Description

Mophun streams are ways of transferring data, for across networks, wireless links or files.

Mophun supports a large number of stream types, but not all implementations provide support for all types of streams. For example, bluetooth may be missing in some implementations.

5.42.2 Define Documentation

5.42.2.1 #define ntohl(netlong) htonl(netlong)

Convert from network-endian (big-endian) to host-endian.

5.42.2.2 #define ntohs(netshort) htons(netshort)

Convert from network-endian (big-endian) to host-endian.

5.42.2.3 #define vStreamReadFrom vStreamFrom

Alias for vStreamFrom().

5.42.2.4 #define vStreamWriteTo vStreamTo

Alias for vStreamTo().

5.42.3 Function Documentation

5.42.3.1 unsigned long htonl (unsigned long *hostlong*)

Convert from host-endian to network-endian (big-endian).

5.42.3.2 unsigned short htons (unsigned short *hostshort*)

Convert from host-endian to network-endian (big-endian).

5.42.3.3 void vStreamClose (int32_t *handle*)

Close a stream.

This function closes a stream that was opened by vStreamOpen().

Parameters:

handle A handle to the stream to close.

See also:

vStreamOpen

5.42.3.4 int32_t vStreamFrom (int32_t *handle*, void * *buf*, int32_t *count*, void * *args*)

Read from stream with attributes.

This function reads data from a stream using the stream-type specific attributes specified in `args`.

Parameters:

handle Stream handle from which data will be read.

buf Pointer to a buffer into which data will be copied.

count The maximum number of bytes to read.

args Stream-type specific arguments, if NULL the call is identical to vStreamRead().

Returns:

The number of bytes read, -1 on error. If zero is returned

See also:

vStreamRead, vStreamTo, vStreamWrite.

5.42.3.5 int32_t vStreamMode (int32_t *handle*)

Get stream type and mode.

This function returns the type and mode of a stream as was specified when the stream was opened.

Parameters:

handle A stream handle.

Returns:

The stream type and mode.

5.42.3.6 int32_t vStreamOpen (const char * *name*, int32_t *mode*)

Create a stream.

This function creates a stream and returns a handle to it which may be used to read and write data to the stream.

Parameters:

name The identifier of the stream, it is interpreted differently depending on the stream type.

mode The type and mode of stream. The stream type is combined with the stream flags, see Stream types and Stream Modes.

Stream type	Meaning of name
`STREAM_FILE`	A filename. If no path is specified the directory of the game is assumed. Note that it may not be possible to open files outside the games directory.
`STREAM_TCP and STREAM_UDP`	A dotted decimal IP address, a hostname or NULL if any host is accepted
`STREAM_BLUETOOTH`	NULL to let the user select a bluetooth device in a list, the name of a bluetooth device or a hexadecimal bluetooth address (each byte separated by colons).
`STREAM_IR`	Ignored
`STREAM_SMS`	The name is divided into two components: `TELNUM[:[:]NAME]`. The first part is the telephone number that messages will be sent to, the second part is an identifier of the game. If no telephone number is specified the stream is put into receiving mode (`STREAM_ACCEPT` must also be specified). See SMS streams for details.
`STREAM_RESOURCE`	NULL to open an internal resource. In API version 1.50 and later it is possible to specify the filename of an external resource file.
`STREAM_HTTP`	An HTTP URL. Only available in API version 1.50 and later. See HTTP streams for details.

Remarks:

5.42.3.7 int32_t vStreamRead (int32_t *handle*, void * *buf*, int32_t *count*)

Read data from a stream.

This function reads data from the stream into a caller-supplied buffer. For network and wireless oriented streams, this is a blocking call, the function does not return until there is data available on the stream or an error occurs.

Parameters:

handle Stream handle from which data will be read.

buf Pointer to a buffer into which data will be copied.

count The maximum number of bytes to read.

Returns:

The number of bytes read, -1 on error. If zero is returned it indicates an end-of-file condition, for network oriented streams this indicates that the connection was closed by the other side.

See also:

vStreamFrom, vStreamWrite.

5.42.3.8 int32_t vStreamReady (int32_t *handle*, int32_t *mode*)

Check stream state.

This function is used to determine whether a stream is ready to read or write data. For network and wireless oriented streams it also indicates if the connection has been closed.

Parameters:

handle A stream handle.

mode One or both of `STREAM_READ` and `STREAM_WRITE`.

Returns:

A bitmask indicating the state of the stream.

Remarks:

For network and wireless oriented streams the function returns STREAM_READ in the case the connection has been closed, a subsequent call to vStreamRead() will return 0 to indicate the closed connection.

Example:

```
int readyFlags = vStreamReady(handle, STREAM_READ | STREAM_WRITE);
if (readyFlags & STREAM_READ)
  ... // Stream is ready for reading
if (readyFlags & STREAM_WRITE)
  ... // Stream is ready for writing
```

5.42.3.9 int32_t vStreamSeek (int32_t *handle*, int32_t *where*, int32_t *whence*)

Seek in stream.

This function updates the current in a stream. It is currently only supported for file and resource streams.

Parameters:

handle A stream handle.

where Offset to add to the seek reference point.

whence The seek reference point, see Seek Modes.

Returns:
The new position (relative to start) in the stream, -1 on error.

It is possible to figure out the current position in a stream by calling the function as follows:

```
// Get current position in stream
int32_t curPos = vStreamSeek(handle, 0, VSEEK_CUR);
```

It is possible to determine the length of a file or resource by calling vStreamSeek as in the following example:

```
// Get number of bytes in stream
int32_t streamLen = vStreamSeek(handle, 0, VSEEK_END);
```

5.42.3.10 int32_t vStreamTo (int32_t *handle*, void * *buf*, int32_t *count*, void * *args*)

Write to stream with attributes.

This function writes data to a stream using the stream-type specific attributes specified in `args`.

Parameters:
handle Stream handle to which data will be written.

buf Pointer to a buffer from which data will be written.

count The number of bytes to write.

args Stream-type specific arguments, if NULL the call is identical to vStreamWrite().

Returns:
The number of bytes written, -1 on error.

See also:
vStreamWrite, vStreamFrom, vStreamRead.

5.42.3.11 int32_t vStreamWrite (int32_t *handle*, const void * *buf*, int32_t *count*)

Write data to a stream.

This function writes data from a caller-supplied buffer to the stream. The function does not return until all data have been handled.

Parameters:
handle Stream handle to which data will be written.

buf Pointer to a buffer from which data will be written.

count The number of bytes to write.

Returns:
The number of bytes written, -1 on error.

See also:
vStreamRead, vStreamTo.

5.43 Stream types

Defines

- #define STREAM_FILE 0
  File-system based stream.
- #define STREAM_TCP 1
  TCP/IP based network stream.
- #define STREAM_UDP 2
  UDP/IP based datagram network stream.
- #define STREAM_BLUETOOTH 3
  BlueTooth RFCOMM stream.
- #define STREAM_BT STREAM_BLUETOOTH
  Alias for STREAM_BLUETOOTH.
- #define STREAM_IR 4
  IrDA wireless link stream.
- #define STREAM_SMS 5
  SMS (Short Message Service) stream.
- #define STREAM_CABLE 6
  Serial cable stream.
- #define STREAM_RESOURCE 7
  Resource stream.
- #define STREAM_HTTP 8
  HTTP stream.

5.43.1 Define Documentation

5.43.1.1 #define STREAM_BLUETOOTH 3

BlueTooth RFCOMM stream.

5.43.1.2 #define STREAM_BT STREAM_BLUETOOTH

Alias for STREAM_BLUETOOTH.

5.43.1.3 #define STREAM_CABLE 6

Serial cable stream.

5.43.1.4 #define STREAM_FILE 0

File-system based stream.

5.43.1.5 #define STREAM_HTTP 8

HTTP stream.

5.43.1.6 #define STREAM_IR 4

IrDA wireless link stream.

5.43.1.7 #define STREAM_RESOURCE 7

Resource stream.

5.43.1.8 #define STREAM_SMS 5

SMS (Short Message Service) stream.

5.43.1.9 #define STREAM_TCP 1

TCP/IP based network stream.

5.43.1.10 #define STREAM_UDP 2

UDP/IP based datagram network stream.

5.44 Stream Modes

Defines

- #define STREAM_READ 0x0100
  Stream supports reading.
- #define STREAM_WRITE 0x0200
  Stream supports writing.
- #define STREAM_READWRITE (STREAM_READ | STREAM_WRITE)
  Stream supports reading and writing.
- #define STREAM_BINARY 0x0400
  Stream is binary (files only).
- #define STREAM_TEXT 0x0000
  Stream is text (files only).
- #define STREAM_CREATE 0x0800
  File is created if it does not exist.
- #define STREAM_TRUNC 0x1000
  File is truncated when opened, STREAM_WRITE must also be specified.
- #define STREAM_EXCL 0x2000
  File creation fails if file already exists.
- #define STREAM_DELETE 0x4000
  File is deleted.
- #define STREAM_ACCEPT 0x8000
  Stream should wait for a client connection.
- #define STREAM_OBEX 0x0400
  Stream should be opened in OBEX mode, only supported on SonyEricsson devices.
- #define STREAM_MODE_MASK 0xFF00
  Mask for the stream mode.
- #define STREAM_PORT_SHIFT 16
  Shift count for network ports.

5.44.1 Define Documentation

5.44.1.1 #define STREAM_ACCEPT 0x8000

Stream should wait for a client connection.

5.44.1.2 #define STREAM_BINARY 0x0400

Stream is binary (files only).

5.44.1.3 #define STREAM_CREATE 0x0800

File is created if it does not exist.

5.44.1.4 #define STREAM_DELETE 0x4000

File is deleted.

5.44.1.5 #define STREAM_EXCL 0x2000

File creation fails if file already exists.

5.44.1.6 #define STREAM_MODE_MASK 0xFF00

Mask for the stream mode.

5.44.1.7 #define STREAM_OBEX 0x0400

Stream should be opened in OBEX mode, only supported on SonyEricsson devices.

5.44.1.8 #define STREAM_PORT_SHIFT 16

Shift count for network ports.

5.44.1.9 #define STREAM_READ 0x0100

Stream supports reading.

5.44.1.10 #define STREAM_READWRITE (STREAM_READ | STREAM_WRITE)

Stream supports reading and writing.

5.44.1.11 #define STREAM_TEXT 0x0000

Stream is text (files only).

5.44.1.12 #define STREAM_TRUNC 0x1000

File is truncated when opened, STREAM_WRITE must also be specified.

5.44.1.13 #define STREAM_WRITE 0x0200

Stream supports writing.

5.45 Seek Modes

Defines

- #define VSEEK_SET 0

  Seek is relative to start of the stream.

- #define VSEEK_CUR 1

  Seek is relative to current position in stream.

- #define VSEEK_END 2

  Seek is relative to end of the stream.

5.45.1 Define Documentation

5.45.1.1 #define VSEEK_CUR 1

Seek is relative to current position in stream.

5.45.1.2 #define VSEEK_END 2

Seek is relative to end of the stream.

5.45.1.3 #define VSEEK_SET 0

Seek is relative to start of the stream.

5.46 File helper macros

Mophun defines a number of file macros that may be used instead of the low-level stream functions.

Defines

- #define vFileOpen(filename, mode) vStreamOpen(filename, mode)

  Open existing file.

- #define vFileCreate(filename) vStreamOpen(filename, STREAM_WRITE | STREAM_CREATE | STREAM_TRUNC)

  Create and truncate file.

- #define vFileClose(h) vStreamClose(h)

  Close file.

- #define vFileRead(fd, buf, count) vStreamRead(fd,buf,count)

  Read from file.

- #define vFileWrite(fd, buf, count) vStreamWrite(fd,buf,count)

  Write to file.

- #define vFileSeek(fd, ofs, whence) vStreamSeek(fd, ofs, whence)

  Seek in file.

- #define vFileDelete(filename) vStreamOpen(filename, STREAM_DELETE)

  Delete file.

5.46.1 Detailed Description

Mophun defines a number of file macros that may be used instead of the low-level stream functions.

5.46.2 Define Documentation

5.46.2.1 #define vFileClose(h) vStreamClose(h)

Close file.

5.46.2.2 #define vFileCreate(filename) vStreamOpen(filename, STREAM_WRITE | STREAM_CREATE | STREAM_TRUNC)

Create and truncate file.

5.46.2.3 #define vFileDelete(filename) vStreamOpen(filename, STREAM_DELETE)

Delete file.

This function deletes the specified file.

Parameters:
filename The name of the file to delete.

Returns:
0 If the file was deleted, -1 on error.

Since:
API version 1.50.

5.46.2.4 #define vFileOpen(filename, mode) vStreamOpen(filename, mode)

Open existing file.

5.46.2.5 #define vFileRead(fd, buf, count) vStreamRead(fd,buf,count)

Read from file.

5.46.2.6 #define vFileSeek(fd, ofs, whence) vStreamSeek(fd, ofs, whence)

Seek in file.

5.46.2.7 #define vFileWrite(fd, buf, count) vStreamWrite(fd,buf,count)

Write to file.

5.47 TCP macros

Mophun defines a number of macros that may be used for establishing TCP connections instead of the low-level stream functions.

Defines

- #define vStreamConnect(name, port) vStreamOpen(name, STREAM_TCP | STREAM_READWRITE | ((port) << STREAM_PORT_SHIFT))

  Connect to a host.

- #define vStreamAccept(port) vStreamOpen(NULL, STREAM_TCP | STREAM_READWRITE | STREAM_ACCEPT | ((port) << STREAM_PORT_SHIFT))

  Accept TCP connections.

5.47.1 Detailed Description

Mophun defines a number of macros that may be used for establishing TCP connections instead of the low-level stream functions.

5.47.2 Define Documentation

5.47.2.1 #define vStreamAccept(port) vStreamOpen(NULL, STREAM_TCP | STREAM_READWRITE | STREAM_ACCEPT | ((port) << STREAM_PORT_SHIFT))

Accept TCP connections.

Parameters:

port The port to accept connections on.

Returns:

A stream handle when a connection is established, -1 on error.

5.47.2.2 #define vStreamConnect(name, port) vStreamOpen(name, STREAM_TCP | STREAM_READWRITE | ((port) << STREAM_PORT_SHIFT))

Connect to a host.

Parameters:

name Dotted-decimal IP address or hostname.

port The port to connect to.

Returns:

A stream handle, -1 on error.

5.48 Resource macros

Mophun defines a number of macros that may be used to handle resources instead of using the low-level stream functions.

Defines

- #define vResOpen(filename, resid) vStreamOpen(filename, STREAM_READ|STREAM_-RESOURCE | ((resid) << 16))

  Open resource for reading.

- #define vResOpenMode(filename, resid, mode) vStreamOpen(filename, (mode)|STREAM_-RESOURCE | ((resid) << 16))

  Open resource with specific mode.

- #define vResClose(fd) vStreamClose(fd)

  Close resource.

- #define vResRead(fd, buf, count) vStreamRead(fd, buf, count)

  Read data from resource.

- #define vResWrite(fd, buf, count) vStreamWrite(fd, buf, count)

  Write data to resource.

- #define vResSeek(fd, ofs, whence) vStreamSeek(fd, ofs, whence)

  Seek within resource.

5.48.1 Detailed Description

Mophun defines a number of macros that may be used to handle resources instead of using the low-level stream functions.

5.48.2 Define Documentation

5.48.2.1 #define vResClose(fd) vStreamClose(fd)

Close resource.

5.48.2.2 #define vResOpen(filename, resid) vStreamOpen(filename, STREAM_READ|STREAM_RESOURCE | ((resid) << 16))

Open resource for reading.

Parameters:

filename Name of the resource file, NULL to use internal resources.

resid The resource identifier.

Returns:
A stream handle, -1 on error.

Since:
External resource files are only supported in API versions 1.50 and later.

5.48.2.3 #define vResOpenMode(filename, resid, mode) vStreamOpen(filename, (mode)|STREAM_RESOURCE | ((resid) << 16))

Open resource with specific mode.

Parameters:
filename Name of the resource file, NULL to use internal resources.

resid The resource identifier.

mode The stream mode, see Stream Modes.

Returns:
A stream handle, -1 on error.

Since:
External resource files are only supported in API versions 1.50 and later.

5.48.2.4 #define vResRead(fd, buf, count) vStreamRead(fd, buf, count)

Read data from resource.

5.48.2.5 #define vResSeek(fd, ofs, whence) vStreamSeek(fd, ofs, whence)

Seek within resource.

5.48.2.6 #define vResWrite(fd, buf, count) vStreamWrite(fd, buf, count)

Write data to resource.

5.49 SMS streams

SMS streams send and recieve SMS messages. SMS streams are special in the sense that they are not really a stream of bytes and they are do not have an established connection between the end-points of the stream.

SMS streams read and write single messages for each call to vStreamRead() or vStreamWrite().

The identifier part of the name parameter, to the vStreamOpen() call, is used as a prefix that is inserted at the start of each message sent: `BEGIN:EGAMEmvm/NAME:[MESSAGE]`.

Sms sending example:

```
#include <vmgp.h>
#include <vstream.h>

#define SMS_TAG_NAME ":sms"
#define NUMBER "+46707360278"

char strPhone[] = NUMBER;
char strPhoneNrAndTag[45];
char strBody[] = "Testing sms!";

int main()
{
        int fd;

        vStrCpy(strPhoneNrAndTag,strPhone);
        vStrCpy(strPhoneNrAndTag + vStrLen(strPhone), SMS_TAG_NAME);

        fd = vStreamOpen(strPhoneNrAndTag, STREAM_SMS | STREAM_WRITE);
        vStreamWrite(fd, strBody, vStrLen(strBody));

        vStreamClose(fd);
        return 0;
}
```

Sms receiving example:

```
#include <vmgp.h>
#include <vstream.h>

#define SMS_TAG_NAME ":sms"

char smsBuf[160];

int main()
{
        int fd = vStreamOpen(SMS_TAG_NAME, STREAM_SMS | STREAM_READ | STREAM_ACCEPT);
        vStreamRead(fd, smsBuf, sizeof(smsBuf));
        vStreamClose(fd);
}
```

Remarks:

Since API version 1.50 it is possible to use double colons to define the whole prefix inserted into messages. For example specifying `+467012345678::MYPFX` will insert `MYPFX:` as a prefix in all messages sent, and incoming messages with the specified prefix are may be received my the game.

5.50 Data certificate library

The data certificate library is used to verify data certificates.

Modules

- Data certficate resources

  Games may include datacert.h in their resource files to get access to a number of macros used to define data certificates in resources.

Return values

- #define DATACERT_FAILURE 0

  Certificate is not valid.

- #define DATACERT_SUCCESS 1

  Certificate is valid.

- #define DATACERT_EXPIRED 2

  Certificate has expired.

Functions

- int32_t vlDataCertCheck (uint8_t *cert)

  Check data certificate in memory.

- int32_t vlDataCertCheckStream (int handle, int *ptagsize, int *pdatasize, int *pdataofs)

  Check data certificate from stream.

- int32_t vlDataCertGetTagDataSize (uint8_t *cert)

  Get size of certificate tags.

- int32_t vlDataCertGetTagStart (void)

  Get offset to certificate tags.

- uint32_t vlDataCertGetCertID (uint8_t *cert)

  Get certificate ID.

- int32_t vlDataCertCheckFile (const char *filename)

  Check data certificate in file.

- int32_t vlDataCertCheckResource (int res)

  Check data certificate in resource.

- int32_t vlDataCertCheckFile2 (const char *filename, int *ptagsize, int *pdatasize, int *pdataofs)

  Check data certificate in file and return information.

- int32_t vlDataCertCheckResource2 (int res, int *ptagsize, int *pdatasize, int *pdataofs)

  Check data certificate in resource.

5.50.1 Detailed Description

The data certificate library is used to verify data certificates.

This allow a game to verify that the player of the game is authorized.

5.50.2 Certificate tags

A certificate can contain tags that contain information used by a game. Tags are stored in a null-terminated string. Each tag is separated by a colon. Mophun defines a number of standard tags that all start with 'M', custom tags must not start with 'M'.

5.50.2.1 Standard tags

The following tags are verified by all mophun games:

- `M001` followed by a date indicates the starting date when the game becomes runnable.
- `M002` followed by a date indicating the end date of a "first time start-up interval".
- `M003` followed by an expiration date.
- `M010` followed by a value uniquely identifying the end user (32 characters max).

\ M001 and `M002` can be used together for a "first time start-up interval" feature: the very first time the game is started on a phone must be within that time period. After this has been done it never expires. Note: level files don't use `M002` (only .mpn files and empty .mpc files).

M001 and M003 can be used together to support a subscription based model where the game can only run during that specific time. The difference from the time bomb feature is that it is verified every time the game is started.

All the dates are inclusive and are formatted as YYYYMMDD. For example a tag with first time start-up interval from Jan 1:st to 31:st 2003 with the subscriber nr 999999 would be: `M00120030101`:M00220030131:M010999999

5.50.2.2 Custom tags

Up to 160 bytes can be used for game specific tags as long as they don't start with 'M'.

5.50.3 Define Documentation

5.50.3.1 #define DATACERT_EXPIRED 2

Certificate has expired.

5.50.3.2 #define DATACERT_FAILURE 0

Certificate is not valid.

5.50.3.3 #define DATACERT_SUCCESS 1

Certificate is valid.

5.50.4 Function Documentation

5.50.4.1 int32_t vlDataCertCheck (uint8_t * *cert*)

Check data certificate in memory.

This function verifies a data certificate stored in memory.

Parameters:
cert Pointer to data certificate.

Returns:
See Return values.

5.50.4.2 int32_t vlDataCertCheckFile (const char * *filename*)

Check data certificate in file.

This function verifies a data certificate in the specified file.

Parameters:
filename Name of the file that contains a data certificate.

Returns:
See Return values

5.50.4.3 int32_t vlDataCertCheckFile2 (const char * *filename*, int * *ptagsize*, int * *pdatasize*, int * *pdataofs*)

Check data certificate in file and return information.

This function verifies a data certificate in the specified file and returns information about the certificate.

Parameters:
filename Name of the file that contains a data certificate.

ptagsize If not NULL, it will be set to the size of the tags on return.

pdatasize If not NULL, it will be set to the size of the certificate data on return.

pdataofs If not NULL, it will be set to the offset to the certificate data (relative to start of stream).

Returns:
See Return values

Example:

```
int tagsize, datasize, dataofs;
int result = vlDataCertCheckFile2("cert.mpc", &tagsize, &datasize, &dataofs);
```

5.50.4.4 int32_t vlDataCertCheckResource (int *res*)

Check data certificate in resource.

This function verifies a data certificate in the specified resource.

Parameters:

res Resource index that contains a data certificate.

Returns:

See Return values

Example:

```
int result = vlDataCertCheckResource(DATACERT);
```

5.50.4.5 int32_t vlDataCertCheckResource2 (int *res*, int * *ptagsize*, int * *pdatasize*, int * *pdataofs*)

Check data certificate in resource.

This function verifies a data certificate in the specified resource.and returns information about the certificate.

Parameters:

res Resource index that contains a data certificate.

ptagsize If not NULL, it will be set to the size of the tags on return.

pdatasize If not NULL, it will be set to the size of the certificate data on return.

pdataofs If not NULL, it will be set to the offset to the certificate data (relative to start of stream).

Returns:

See Return values

Example:

```
int tagsize, datasize, dataofs;
int result = vlDataCertCheckResource2(DATACERT, &tagsize, &datasize, &dataofs);
```

5.50.4.6 int32_t vlDataCertCheckStream (int *handle*, int * *ptagsize*, int * *pdatasize*, int * *pdataofs*)

Check data certificate from stream.

This function reads a data certificate from a stream and returns information about the certificate.

Parameters:

handle Handle to a file or resource stream.

ptagsize If not NULL, it will be set to the size of the tags on return.

pdatasize If not NULL, it will be set to the size of the certificate data on return.

pdataofs If not NULL, it will be set to the offset to the certificate data (relative to start of stream).

Returns:
See Return values

Remarks:
If the function returns DATACERT_SUCCESS the current stream position will be at the start of the tags. To get to the data either skip the number of bytes occupied by the tags or seek to the position returned in `pdataofs`.

Example:

```
int tagsize, datasize, dataofs;
int handle = vStreamOpen("cert.mpc", STREAM_BINARY|STREAM_READ);
int result = vlDataCertCheckStream(handle, &tagsize, &datasize, &dataofs);
if (result == DATACERT_SUCCESS)
{
  // Read data
  vStreamSeek(handle, dataofs, VSEEK_SET);
  vStreamRead(handle, mydata, datasize);
}
vStreamClose(handle);
```

5.50.4.7 uint32_t vlDataCertGetCertID (uint8_t * *cert*)

Get certificate ID.

This function returns the certificate ID if successful, otherwise DATACERT_FAILURE. The certificate ID can be used by a game to see when a new data certificate has been purchased (the game checks a previously stored certificate ID to see if it changes). This is not needed for ordinary pay per level games.

Parameters:
cert Pointer to data certificate.

Returns:
Certificate ID, DATACERT_FAILURE on failure.

5.50.4.8 int32_t vlDataCertGetTagDataSize (uint8_t * *cert*)

Get size of certificate tags.

Parameters:
cert Pointer to data certificate.

Returns:
Size of tags and data in the certificate.

5.50.4.9 int32_t vlDataCertGetTagStart (void)

Get offset to certificate tags.

Returns:
The offset from the start of a data certificate to the certificate tags.

5.51 Data certficate resources

Games may include datacert.h in their resource files to get access to a number of macros used to define data certificates in resources.

Defines

- #define SECURITY_RESOURCE SECURITY DATA { FILL 4,0 }

  Security resource, used internally.

- #define DATACERT_MAGIC 0xee,0x43,0x90,0x72,0x11,0x7F,0x83,0xA5

  Magic number for VendorSigningTool (VST).

- #define DATACERT_MINSIZE 314

  Minimum size of the data certificate resource.

- #define DATACERT_DATA FILL DATACERT_MINSIZE, 0

  Data certificate data.

- #define DATACERT_RESOURCE DATACERT DATA { DATACERT_MAGIC, DATACERT_DATA }

  A complete data certificate resource.

- #define DATACERT_RESOURCE_SIZE(size) DATACERT DATA { DATACERT_MAGIC, FILL size, 0 }

  Helper for custom sized data certificates.

- #define DATACERT_AND_SECURITY_RESOURCE

  Security and data certificate resources.

- #define DATACERT_AND_SECURITY_RESOURCE_SIZE(size)

  Security and data certificate resource of specific.

5.51.1 Detailed Description

Games may include datacert.h in their resource files to get access to a number of macros used to define data certificates in resources.

Example:

```
#include <datacert.h>

// Meta-info should always be first
METAINFO
{
        "Title" : "Super duper game"
}

// Data certificate resource must always follow
// after meta-info.

// Define data cerficicate resource.
```

```
DATACERT_AND_SECURITY_RESOURCE

...
```

5.51.2 Define Documentation

5.51.2.1 #define DATACERT_AND_SECURITY_RESOURCE

Value:

```
SECURITY_RESOURCE \
  DATACERT_RESOURCE
```

Security and data certificate resources.

Use this macro to define a data certificate and a security resource. This macro should be used immediately after the metainfo resource. The resource is called DATACERT.

5.51.2.2 #define DATACERT_AND_SECURITY_RESOURCE_SIZE(size)

Value:

```
SECURITY_RESOURCE \
  DATACERT_RESOURCE_SIZE(size)
```

Security and data certificate resource of specific.

Use this macro to define a data certificate of a specific size and a security resource. This macro should be used immediately after the metainfo resource. The resource is called DATACERT.

5.51.2.3 #define DATACERT_DATA FILL DATACERT_MINSIZE, 0

Data certificate data.

5.51.2.4 #define DATACERT_MAGIC 0xee,0x43,0x90,0x72,0x11,0x7F,0x83,0xA5

Magic number for VendorSigningTool (VST).

5.51.2.5 #define DATACERT_MINSIZE 314

Minimum size of the data certificate resource.

5.51.2.6 #define DATACERT_RESOURCE DATACERT DATA { DATACERT_MAGIC, DATACERT_DATA }

A complete data certificate resource.

Use this macro to define a data certificate. The resource is called DATACERT.

5.51.2.7 #define DATACERT_RESOURCE_SIZE(size) DATACERT DATA { DATACERT_MAGIC, FILL size, 0 }

Helper for custom sized data certificates.

This macro defines a data certificate of a specific size. The resource is called DATACERT.

Parameters:

size The size of the certificate.

5.51.2.8 #define SECURITY_RESOURCE SECURITY DATA { FILL 4,0 }

Security resource, used internally.

Chapter 6

mophun API Data Structure Documentation

6.1 BBOX Struct Reference

Bounding box.

`#include <vmgp3d.h>`

Data Fields

- VECTOR3 min
  Axis aligned min coordinate.
- VECTOR3 max
  Axis aligned max coordinate.

6.1.1 Detailed Description

Bounding box.

6.1.2 Field Documentation

6.1.2.1 VECTOR3 BBOX::max

Axis aligned max coordinate.

6.1.2.2 VECTOR3 BBOX::min

Axis aligned min coordinate.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.2 BEEP Struct Reference

Beep sequence element.

`#include <vmgp.h>`

Data Fields

- uint8_t freq_lo

  Frequency least significant bits.

- uint8_t freq_hi

  Frequency most significant bits.

- uint8_t dur_lo

  Duration least significant bits.

- uint8_t dur_hi

  Duration most significant bits.

- uint8_t vol

  Volume, 0-255.

6.2.1 Detailed Description

Beep sequence element.

This structure defines tone in a sequence of tones.

6.2.2 Field Documentation

6.2.2.1 uint8_t BEEP::dur_hi

Duration most significant bits.

6.2.2.2 uint8_t BEEP::dur_lo

Duration least significant bits.

6.2.2.3 uint8_t BEEP::freq_hi

Frequency most significant bits.

6.2.2.4 uint8_t BEEP::freq_lo

Frequency least significant bits.

6.2.2.5 uint8_t BEEP::vol

Volume, 0-255.

The documentation for this struct was generated from the following file:

- vmgp.h

6.3 CAPS Union Reference

Union of all capability structures.

`#include <vmgpcaps.h>`

Data Fields

- CAPSHDR **hdr**
- INPUTCAPS **input**
- VIDEOCAPS **video**
- COMMCAPS **comm**
- SOUNDCAPS **sound**
- SYSCAPS **sys**

6.3.1 Detailed Description

Union of all capability structures.

The documentation for this union was generated from the following file:

- vmgpcaps.h

6.4 COMMCAPS Struct Reference

Communication capabilities.

`#include <vmgpcaps.h>`

Data Fields

- uint16_t size

  Size of the structure.

- uint16_t flags

  Communication flags.

6.4.1 Detailed Description

Communication capabilities.

This structure contains information about the communication capabilities of the device.

6.4.2 Field Documentation

6.4.2.1 uint16_t COMMCAPS::flags

Communication flags.

See CommFlags for a list of possible values.

6.4.2.2 uint16_t COMMCAPS::size

Size of the structure.

Must be at least `sizeof(COMMCAPS)`.

The documentation for this struct was generated from the following file:

- vmgpcaps.h

6.5 COMPRESSEDFILE Struct Reference

Compressed data header.

`#include <vmgp.h>`

Data Fields

- uint8_t **cnt**
- uint8_t **offset**
- uint16_t **crc16**
- uint16_t **version**
- uint16_t **option**
- uint32_t **srcsize**
- uint32_t **dstsize**
- uint32_t **literalsize**

6.5.1 Detailed Description

Compressed data header.

The documentation for this struct was generated from the following file:

- vmgp.h

6.6 DCOLOR Struct Reference

Diffuse color.

`#include <vmgp3d.h>`

Data Fields

- uint8_t b

  Blue 8-bit diffuse component.

- uint8_t g

  Green 8-bit diffuse component.

- uint8_t r

  Red 8-bit diffuse component.

- uint8_t a

  Alpha 8-bit component.

6.6.1 Detailed Description

Diffuse color.

This structure defines diffuse color components.

6.6.2 Field Documentation

6.6.2.1 uint8_t DCOLOR::a

Alpha 8-bit component.

6.6.2.2 uint8_t DCOLOR::b

Blue 8-bit diffuse component.

6.6.2.3 uint8_t DCOLOR::g

Green 8-bit diffuse component.

6.6.2.4 uint8_t DCOLOR::r

Red 8-bit diffuse component.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.7 DSCOLOR Struct Reference

Diffuse and specular color.

`#include <vmgp3d.h>`

Data Fields

- DCOLOR diff

  Diffuse color components.

- SCOLOR spec

  Specular color components.

6.7.1 Detailed Description

Diffuse and specular color.

This structure defines diffuse and specular color components.

6.7.2 Field Documentation

6.7.2.1 DCOLOR DSCOLOR::diff

Diffuse color components.

6.7.2.2 SCOLOR DSCOLOR::spec

Specular color components.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.8 HTTP_HEADERS Struct Reference

HTTP header struct.

`#include <vhttp.h>`

Public Member Functions

- vPtr (const char, headers)

  The headers.

Data Fields

- uint32_t size

  Size of the structure.

- uint32_t method

  The HTTP stream command.

6.8.1 Detailed Description

HTTP header struct.

This structure is used to specify or extract HTTP headers in an HTTP request.

6.8.2 Member Function Documentation

6.8.2.1 HTTP_HEADERS::vPtr (const *char*, headers)

The headers.

The headers are direct HTTP headers in a null-terminated string. Headers are separated by CRLF pairs.

6.8.3 Field Documentation

6.8.3.1 uint32_t HTTP_HEADERS::method

The HTTP stream command.

Must be `STREAM_HTTP_HEADERS`. See HTTP stream commands..

6.8.3.2 uint32_t HTTP_HEADERS::size

Size of the structure.

The structure size must be `sizeof(HTTP_HEADERS)`.

The documentation for this struct was generated from the following file:

- vhttp.h

6.9 HTTP_RESULT Struct Reference

HTTP request result.

`#include <vhttp.h>`

Data Fields

- uint32_t size

  Size of the structure.

- uint32_t method

  The HTTP stream command.

- uint32_t code

  The HTTP status code.

- uint32_t contentlength

  The length of thee HTTP response.

6.9.1 Detailed Description

HTTP request result.

This structure is used to get the result of an HTTP request.

6.9.2 Field Documentation

6.9.2.1 uint32_t HTTP_RESULT::code

The HTTP status code.

See THTTPStatusCode for a list of possible HTTP status codes.

6.9.2.2 uint32_t HTTP_RESULT::contentlength

The length of thee HTTP response.

This member is set to the length of the HTTP response body. If the length is not available it is set to zero.

6.9.2.3 uint32_t HTTP_RESULT::method

The HTTP stream command.

Must be `STREAM_HTTP_RESULT`. See HTTP stream commands..

6.9.2.4 uint32_t HTTP_RESULT::size

Size of the structure.

The structure size must be `sizeof(HTTP_RESULT)`.

The documentation for this struct was generated from the following file:

- vhttp.h

6.10 INPUTCAPS Struct Reference

Input capabilities.

`#include <vmgpcaps.h>`

Data Fields

- uint16_t size

  Size of the structure.

- uint16_t flags

  Input flags, see InputFlags.

- uint8_t keycount

  The number of physical keys.

6.10.1 Detailed Description

Input capabilities.

This structure contains information about the input capabilities of the device.

6.10.2 Field Documentation

6.10.2.1 uint16_t INPUTCAPS::flags

Input flags, see InputFlags.

6.10.2.2 uint8_t INPUTCAPS::keycount

The number of physical keys.

6.10.2.3 uint16_t INPUTCAPS::size

Size of the structure.

Must be at least `sizeof(INPUTCAPS)`.

The documentation for this struct was generated from the following file:

- vmgpcaps.h

6.11 MAP_HEADER Struct Reference

Map definition.

`#include <vmgp.h>`

Data Fields

- uint8_t flag

  Tilemap flags, see MapAttr.

- uint8_t format

  Tile graphics format, VCAPS_IND2-VCAPS_RGB332, see Graphics formats.

- uint8_t width

  Number of tiles in horizontal direction.

- uint8_t height

  Number of tiles in vertical direction.

- uint8_t animationspeed

  Number of frames between animations.

- uint8_t animationcount

  Internal.

- uint8_t animationactive

  Internal.

- uint8_t pad

  Padding.

- uint16_t xpan

  Number of pixel to pan map window in horizontal direction relative to screen window.

- uint16_t ypan

  Number of pixel to pan map window in vertical direction relative to screen window.

- uint16_t x

  Horizontal pixel position relative to mapstart.

- uint16_t y

  Vertical pixel position relative to mapstart.

- uint8_t * mapoffset

  Pointer to tile map.

- uint8_t * tiledata

  Pointer to linear tile data.

6.11.1 Detailed Description

Map definition.

This structure defines the properties of a tilemap layer.

6.11.2 Field Documentation

6.11.2.1 uint8_t MAP_HEADER::animationactive

Internal.

6.11.2.2 uint8_t MAP_HEADER::animationcount

Internal.

6.11.2.3 uint8_t MAP_HEADER::animationspeed

Number of frames between animations.

If api version 1.30 or newer is used, 0 is treated as animation each frame. Otherwise 0 has the same animation speed as 1, namely every second frame. See section API Version about setting api version.

6.11.2.4 uint8_t MAP_HEADER::flag

Tilemap flags, see MapAttr.

6.11.2.5 uint8_t MAP_HEADER::format

Tile graphics format, VCAPS_IND2-VCAPS_RGB332, see Graphics formats.

6.11.2.6 uint8_t MAP_HEADER::height

Number of tiles in vertical direction.

6.11.2.7 uint8_t* MAP_HEADER::mapoffset

Pointer to tile map.

Every tile is 8-bit. If attribute is used, it will be located between tiles and is also 8-bit. Tile no zero is not drawn and tile number one is pointed to start of tiledata, so you are only able to use 255 tiles.

6.11.2.8 uint8_t MAP_HEADER::pad

Padding.

6.11.2.9 uint8_t* MAP_HEADER::tiledata

Pointer to linear tile data.

6.11.2.10 uint8_t MAP_HEADER::width

Number of tiles in horizontal direction.

6.11.2.11 uint16_t MAP_HEADER::x

Horizontal pixel position relative to mapstart.

6.11.2.12 uint16_t MAP_HEADER::xpan

Number of pixel to pan map window in horizontal direction relative to screen window.

6.11.2.13 uint16_t MAP_HEADER::y

Vertical pixel position relative to mapstart.

6.11.2.14 uint16_t MAP_HEADER::ypan

Number of pixel to pan map window in vertical direction relative to screen window.

The documentation for this struct was generated from the following file:

- vmgp.h

6.12 MATRIX Struct Reference

Matrix type.

`#include <vmgp3d.h>`

Data Fields

- fixed32_t m [4][4]

  Rows and columns of matrix.

6.12.1 Detailed Description

Matrix type.

This structure defines a matrix.

6.12.2 Field Documentation

6.12.2.1 fixed32_t MATRIX::m[4][4]

Rows and columns of matrix.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.13 PLANE Struct Reference

Plane.

`#include <vmgp3d.h>`

Data Fields

- VECTOR3 n

  Normal vector.

- fixed32_t d

  Fixed point distance to plane.

6.13.1 Detailed Description

Plane.

6.13.2 Field Documentation

6.13.2.1 fixed32_t PLANE::d

Fixed point distance to plane.

6.13.2.2 VECTOR3 PLANE::n

Normal vector.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.14 SCOLOR Struct Reference

Specular color.

`#include <vmgp3d.h>`

Data Fields

- uint8_t b

  Blue 8-bit specular component.

- uint8_t g

  Green 8-bit specular component.

- uint8_t r

  Red 8-bit specular component.

- uint8_t f

  Fog 8-bit component.

6.14.1 Detailed Description

Specular color.

This structure defines specular color components.

6.14.2 Field Documentation

6.14.2.1 uint8_t SCOLOR::b

Blue 8-bit specular component.

6.14.2.2 uint8_t SCOLOR::f

Fog 8-bit component.

6.14.2.3 uint8_t SCOLOR::g

Green 8-bit specular component.

6.14.2.4 uint8_t SCOLOR::r

Red 8-bit specular component.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.15 SOUNDCAPS Struct Reference

Sound capabilities.

`#include <vmgpcaps.h>`

Data Fields

- uint16_t size
  Size of the structure.
- uint16_t flags
  Sound flags.
- SOUNDCONFIG config
  PCM output settings.

6.15.1 Detailed Description

Sound capabilities.

This structure contains information about the sound capabilities of the device.

6.15.2 Field Documentation

6.15.2.1 SOUNDCONFIG SOUNDCAPS::config

PCM output settings.

Since:
API version 1.50

6.15.2.2 uint16_t SOUNDCAPS::flags

Sound flags.

See SoundFlags for list of possible values.

6.15.2.3 uint16_t SOUNDCAPS::size

Size of the structure.

Must be at least `sizeof(SOUNDCAPS)`.

The documentation for this struct was generated from the following file:

- vmgpcaps.h

6.16 SOUNDCONFIG Struct Reference

PCM output settings.

`#include <vmgpcaps.h>`

Data Fields

- uint16_t sampleFrequency
  The sample frequency (samplerate).
- uint16_t numChannels
  Number of output channels.
- uint16_t bitsPerSample
  Bits per sample.
- uint16_t numMixerChannels
  Number of mixer channels.

6.16.1 Detailed Description

PCM output settings.

This structure contains the settings for the PCM output device.

See also:
SoundApi.

Since:
API version 1.50

6.16.2 Field Documentation

6.16.2.1 uint16_t SOUNDCONFIG::bitsPerSample

Bits per sample.

8 or 16.

6.16.2.2 uint16_t SOUNDCONFIG::numChannels

Number of output channels.

1 == mono output, 2 == stereo output

6.16.2.3 uint16_t SOUNDCONFIG::numMixerChannels

Number of mixer channels.

This is the maximum number simultaneously playing sounds.

6.16.2.4 uint16_t SOUNDCONFIG::sampleFrequency

The sample frequency (samplerate).

For example: 8000, 11025, 1600, 22050, 44100.

The documentation for this struct was generated from the following file:

- vmgpcaps.h

6.17 SPRITE Struct Reference

Sprite header.

`#include <vmgp.h>`

Data Fields

- uint8_t palindex

  Palette offset, only used by IND2, IND4 and IND16 formats.

- uint8_t format

  The format of the sprite pixel data.

- int16_t centerx

  The horizontal center of the sprite.

- int16_t centery

  The vertical center of the sprite.

- uint16_t width

  The width of the sprite.

- uint16_t height

  The height of the sprite.

6.17.1 Detailed Description

Sprite header.

A SPRITE structure describes a sprite object, which is drawn by vDrawObject(), vUpdateSprite(void) and vUpdateSpriteMap(). The sprite pixel data immediately follows after the structure. See also Graphics formats.

Remarks:

Supported formats and sprite widths:

- VCAPS_IND2
  8,16,24,32
- VCAPS_IND4
  4,8,12,16
- VCAPS_IND16
  2,4,6,8
- VCAPS_IND256
  1,2,3,4
- VCAPS_RGB332
  1,2,3,4

6.17.2 Field Documentation

6.17.2.1 int16_t SPRITE::centerx

The horizontal center of the sprite.

6.17.2.2 int16_t SPRITE::centery

The vertical center of the sprite.

6.17.2.3 uint8_t SPRITE::format

The format of the sprite pixel data.

Currently only formats VCAPS_IND2 to VCAPS_RGB332 are supported.

6.17.2.4 uint16_t SPRITE::height

The height of the sprite.

6.17.2.5 uint8_t SPRITE::palindex

Palette offset, only used by IND2, IND4 and IND16 formats.

6.17.2.6 uint16_t SPRITE::width

The width of the sprite.

The documentation for this struct was generated from the following file:

- vmgp.h

6.18 SYSCAPS Struct Reference

System capabilities.

`#include <vmgpcaps.h>`

Data Fields

- uint16_t size
  Size of the structure.
- uint16_t flags
  System capability flags, see SysCapsFlags.
- uint32_t id
  Device id, see MAKE_DEVID.
- uint32_t vendorflags
  Vendor specific flags.

6.18.1 Detailed Description

System capabilities.

This capability structure contains information about the system, such as device vendor and model.

6.18.2 Field Documentation

6.18.2.1 uint16_t SYSCAPS::flags

System capability flags, see SysCapsFlags.

6.18.2.2 uint32_t SYSCAPS::id

Device id, see MAKE_DEVID.

6.18.2.3 uint16_t SYSCAPS::size

Size of the structure.

The size must be at least `sizeof(SYSCAPS)`.

6.18.2.4 uint32_t SYSCAPS::vendorflags

Vendor specific flags.

The documentation for this struct was generated from the following file:

- vmgpcaps.h

6.19 TIMEDATE Struct Reference

Time and date.

`#include <vmgp.h>`

Data Fields

- uint16_t year

  The year, i.e 2003.

- uint16_t day

  The day in the month, 1-31.

- uint8_t month

  The month, 1-12.

- uint8_t hour

  The hour, 0-23.

- uint8_t minute

  The minute, 0-59.

- uint8_t second

  The second, 0-59.

6.19.1 Detailed Description

Time and date.

This structure holds date and time in broken down format.

6.19.2 Field Documentation

6.19.2.1 uint16_t TIMEDATE::day

The day in the month, 1-31.

6.19.2.2 uint8_t TIMEDATE::hour

The hour, 0-23.

6.19.2.3 uint8_t TIMEDATE::minute

The minute, 0-59.

6.19.2.4 uint8_t TIMEDATE::month

The month, 1-12.

6.19.2.5 uint8_t TIMEDATE::second

The second, 0-59.

6.19.2.6 uint16_t TIMEDATE::year

The year, i.e 2003.

The documentation for this struct was generated from the following file:

- vmgp.h

6.20 UV Struct Reference

Texture coordinates.

`#include <vmgp3d.h>`

Data Fields

- int16_t u

  Fixed point u coordinate with 9 bit fraction.

- int16_t v

  Fixed point v coordinate with 9 bit fraction.

6.20.1 Detailed Description

Texture coordinates.

6.20.2 Field Documentation

6.20.2.1 int16_t UV::u

Fixed point u coordinate with 9 bit fraction.

6.20.2.2 int16_t UV::v

Fixed point v coordinate with 9 bit fraction.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.21 VDGRAM Struct Reference

UDP datagram address.

`#include <vstream.h>`

Data Fields

- uint32_t size
  Size of the structure.
- uint32_t addr
  Numeric IP address.
- uint32_t port
  IP port.

6.21.1 Detailed Description

UDP datagram address.

This structure specifies the destination or source of an UDP datagram.

6.21.2 Field Documentation

6.21.2.1 uint32_t VDGRAM::addr

Numeric IP address.

6.21.2.2 uint32_t VDGRAM::port

IP port.

6.21.2.3 uint32_t VDGRAM::size

Size of the structure.

The documentation for this struct was generated from the following file:

- vstream.h

6.22 VECTOR3 Struct Reference

3D vector type.

`#include <vmgp3d.h>`

Data Fields

- fixed32_t x
  X coordinate.
- fixed32_t y
  Y coordinate.
- fixed32_t z
  Z coordinate.

6.22.1 Detailed Description

3D vector type.

This structure defines a vector in 3D space.

6.22.2 Field Documentation

6.22.2.1 fixed32_t VECTOR3::x

X coordinate.

6.22.2.2 fixed32_t VECTOR3::y

Y coordinate.

6.22.2.3 fixed32_t VECTOR3::z

Z coordinate.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.23 VECTOR4 Struct Reference

4D vector type.

`#include <vmgp3d.h>`

Data Fields

- fixed32_t x
  X coordinate.
- fixed32_t y
  Y coordinate.
- fixed32_t z
  Z coordinate.
- fixed32_t w
  W coordinate.

6.23.1 Detailed Description

4D vector type.

This structure defines a vector in 4D space.

6.23.2 Field Documentation

6.23.2.1 fixed32_t VECTOR4::w

W coordinate.

6.23.2.2 fixed32_t VECTOR4::x

X coordinate.

6.23.2.3 fixed32_t VECTOR4::y

Y coordinate.

6.23.2.4 fixed32_t VECTOR4::z

Z coordinate.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.24 VERTEX Struct Reference

Vertex.

`#include <vmgp3d.h>`

Data Fields

- VECTOR4 v

  Vector in 4D coordinate space.

- DCOLOR diff

  Diffuse color components.

- SCOLOR spec

  Specular color components.

- UV uv

  Texture coordinates.

6.24.1 Detailed Description

Vertex.

This structure defines a vertex.

6.24.2 Field Documentation

6.24.2.1 DCOLOR VERTEX::diff

Diffuse color components.

6.24.2.2 SCOLOR VERTEX::spec

Specular color components.

6.24.2.3 UV VERTEX::uv

Texture coordinates.

6.24.2.4 VECTOR4 VERTEX::v

Vector in 4D coordinate space.

The documentation for this struct was generated from the following file:

- vmgp3d.h

6.25 VIDEOCAPS Struct Reference

Video capabilities.

`#include <vmgpcaps.h>`

Data Fields

- uint16_t size

  Size of the structure.

- uint16_t flags

  Graphics flags.

- uint16_t width

  The width of the screen in pixels.

- uint16_t height

  The height of the screen in pixels.

6.25.1 Detailed Description

Video capabilities.

This structure contains information about the screen of the device and which graphics API features are available.

6.25.2 Field Documentation

6.25.2.1 uint16_t VIDEOCAPS::flags

Graphics flags.

The least significant 8 bits contain the output graphics format of the screen, see Graphics formats for a list of possible values. For other bits see VCAPS_3D and VCAPS_ORIENTATION.

6.25.2.2 uint16_t VIDEOCAPS::height

The height of the screen in pixels.

6.25.2.3 uint16_t VIDEOCAPS::size

Size of the structure.

Must be at least `sizeof(VIDEOCAPS)`.

6.25.2.4 uint16_t VIDEOCAPS::width

The width of the screen in pixels.

The documentation for this struct was generated from the following file:

- vmgpcaps.h

6.26 VMGPFONT Struct Reference

Font description.

`#include <vmgp.h>`

Data Fields

- uint8_t * fontdata

  Pointer to start of font data.

- uint8_t * chartbl

  Character index table.

- uint8_t bpp

  Number of bits per pixel in font data.

- uint8_t width

  The width in pixels of the font.

- uint8_t height

  The height in pixels of the font.

- uint8_t palindex

  The offset into the palette for non-monochrome fonts.

6.26.1 Detailed Description

Font description.

This structure describes a user-defined font. User-defined fonts may be monochrome or 4-color palette indexed.

The fontdata member points to the font bitmap data. The font bitmap is an array of bits where the least significant bit is the leftmost pixel. Consecutive lines do not have to start on a byte boundary, i.e. the first pixel on a line does not have to be the first bit within a byte. However, each character bitmap must start and end on a byte boundary.

The character table is indexed by character values and contains the number of the character within the font bitmap data. For example if the first supported character in the font is 'A', chartbl['A'] is set to 0 (zero). Characters that do not exist should be set to 0xff.

See also:
vSetActiveFont, vPrint.

6.26.2 Field Documentation

6.26.2.1 uint8_t VMGPFONT::bpp

Number of bits per pixel in font data.

6.26.2.2 uint8_t* VMGPFONT::chartbl

Character index table.

6.26.2.3 uint8_t* VMGPFONT::fontdata

Pointer to start of font data.

6.26.2.4 uint8_t VMGPFONT::height

The height in pixels of the font.

6.26.2.5 uint8_t VMGPFONT::palindex

The offset into the palette for non-monochrome fonts.

6.26.2.6 uint8_t VMGPFONT::width

The width in pixels of the font.

The documentation for this struct was generated from the following file:

- vmgp.h

6.27 VMGPPOLY Struct Reference

Polygon definition.

`#include <vmgp.h>`

Data Fields

- int16_t **x1**
- int16_t **y1**
- int16_t **x2**
- int16_t **y2**
- int16_t **x3**
- int16_t **y3**

6.27.1 Detailed Description

Polygon definition.

This structure is used by vDrawFlatPolygon to draw a filled flat polygon.

The documentation for this struct was generated from the following file:

- vmgp.h

6.28 VMGPRECT Struct Reference

Rectangle.

`#include <vmgp.h>`

Data Fields

- int16_t x
  Left position.
- int16_t y
  Top position.
- uint16_t width
  Width of rectangle.
- uint16_t height
  Height of rectangle.

6.28.1 Detailed Description

Rectangle.

This structure defines a rectangle.

See also:
vSpriteBoxCollision.

6.28.2 Field Documentation

6.28.2.1 uint16_t VMGPRECT::height

Height of rectangle.

6.28.2.2 uint16_t VMGPRECT::width

Width of rectangle.

6.28.2.3 int16_t VMGPRECT::x

Left position.

6.28.2.4 int16_t VMGPRECT::y

Top position.

The documentation for this struct was generated from the following file:

- vmgp.h

6.29 VMGPSTRIDEPTR Struct Reference

Graphics offset description.

`#include <vmgp.h>`

Data Fields

- uint8_t * ptr

  Pointer to start of graphics data, if NULL it is interpreted as (0,0) in the screen.

- uint16_t xpan

  Number of bytes to add to the pointer for the start location.

- uint16_t ypan

  *Number of bytes*stride to add to the pointer for the start location.*

- int16_t stride

  Number of bytes to add to a pointer to get to the next line.

6.29.1 Detailed Description

Graphics offset description.

This structure describes a position in memory with a specified pitch (stride). It is used by vCopyRect() to copy rectangular areas of graphic between two memory locations.

See also:
vCopyRect.

Since:
API version 1.50

6.29.2 Field Documentation

6.29.2.1 uint8_t* VMGPSTRIDEPTR::ptr

Pointer to start of graphics data, if NULL it is interpreted as (0,0) in the screen.

6.29.2.2 int16_t VMGPSTRIDEPTR::stride

Number of bytes to add to a pointer to get to the next line.

6.29.2.3 uint16_t VMGPSTRIDEPTR::xpan

Number of bytes to add to the pointer for the start location.

6.29.2.4 uint16_t VMGPSTRIDEPTR::ypan

Number of bytes∗stride to add to the pointer for the start location.

The documentation for this struct was generated from the following file:

- vmgp.h

Chapter 7

mophun API Page Documentation

7.1 Deprecated List

Global vCheckNetwork(const char *netstr) This function has never been implemented.

Global vSwap(void *ptr, uint32_t n, uint32_t size) Mophun data is always little-endian.

Global vSwap16(uint16_t u16) Mophun data is always little-endian.

Global vSwap32(uint32_t u32) Mophun data is always little-endian.

Index