MOPHUN™ API REFERENCE GUIDE

Version 1.52

20 February 2003

Synergenix Interactive AB
Solna strandväg 96 - Plan 6
17154 Solna - Sweden
info@mophun.com

PREFACE

Overview

This manual documents the mophun™ Application Programmer Interface (API). This manual is a reference that describes the functions that can be called from a mophun™ application. Herein we have dealt with the technical details of the mophun™ API, its methods, data types etc. All information is divided into libraries, each in separate section. The manual is organized into chapters focused on a specific part of the mophun™ environment. Each chapter contains function descriptions for the function relevant to that chapter's subject. Functions do not have a specific order within the chapter. Additionally, each chapter contains definitions of the typedefs used by the functions described within the chapter.

Audience

This manual is intended for application developers who create and/or maintain mophun™ applications for mobile devices using the mophun™ technology. This manual is intended as a reference only.

Related Documents

The following is a list of related mophun™ documentation:

- *mophun™ Programming Guidelines*
- *mophun™ Assembly Reference*
- *mophun™ Resource Compiler*

For the latest versions of these documents see www.mophun.com

CONTENTS

INTRODUCTION

The document is targeted at programmers who develop applications (e.g. games) for mobile devices using the mophun™ technology. Herein we have dealt with the technical details of the mophun™ API, its methods, data types etc. All information is divided into libraries, each with an individual header. Although this document works as a manual, it does not provide a comprehensive overview of the mophun™ technology. For that purpose please read the *mophun™ Programming Guidelines.*

style	description
keyword	A mophun™ method or type that must be typed exactly as shown.
variable	Variable or similar that you can give any name you like.
`code snippet`	A snippet of code example.
`...`	Replace this with your own code.
`#include<vmgp.h>`	The function requires a specific .h-file, in this case vmgp.h.
`Members`	Describes the members of a structure

Some abbreviations:

Abbreviation	Full Name	Description
VMGP	Virtual Mobile Gaming Platform	The internal name for mophun™
VM	Virtual Machine	
PIP	Platform Independent Processor	The name of the mophun™ VM
RTE	Run Time Engine	
BPP	Bits Per Pixel	

DATA TYPES

Primitive data types

char	8 bit signed integer
unsigned char	8 bit unsigned integer
short	16 bit signed integer
unsigned short	16 bit unsigned integer
int	32 bit signed integer
long	32 bit signed integer

Portable data types

Header: `#include <vmgp.h>`

Type	PIP C type
int8_t	char
uint8_t	unsigned char
int16_t	short
uint16_t	unsigned short
int32_t	int
uint32_t	unsigned int
wchar_t	unsigned short

CAPABILITIES

Header: `#include <vmgp.h>`

int32_t vGetCaps(enum CapsQuery *Query*, void **buf*);

- `Query` Specifies the query being made.
- `buf` Pointer to a capability structure. The size member of the structure must be set to the size of the structure before calling `vGetCaps`.

The GetCaps function provides a query interface that lets VMGP programs query the capabilities of the VMGP runtime platform and make decisions to adapt to the environment. The following query types are available:

CAPS_SYSTEM	System Capabilities
CAPS_VIDEO	Video Capabilities
CAPS_INPUT	Input Capabilities
CAPS_SOUND	Sound Capabilities
CAPS_COMM	Communication Capabilities

Returns: Non-zero if successful, zero if request failed.

System capabilities

```
typedef struct {
   uint16_t size;
   uint16_t flags;
   uint32_t id;
   uint32_t vendorflags;
} SYSCAPS;
```

`Members:`

- `size` Must be set to `sizeof (SYSCAPS)`
- `flags` System capabilities flags.
- `id` System identification. The low 16 bits identify the vendor/manufacturer and the high 16 bits identify the individual system, for example a specific terminal.
- `vendorflags` Vendor specific flags.

System capabilities flags	**Description**
SYSTEM_UNICODE	System supports UNICODE
SYSTEM_VIBRATE	System supports vibrator control
SYSTEM_BIGENDIAN	The system is big endian.

uint16_t DEVICE_VENDOR`(uint32_t id);`
Return the vendor part of a system id.

Possible values are:

```
#define VENDOR_UNKNOWN       0
#define VENDOR_SYNERGENIX    1
#define VENDOR_PALM          2
#define VENDOR_SONYERICSSON  3
```

uint16_t DEVICE_NUMBER(uint32_t id**);**
Return the vendor specific system number of a system id.

Possible values when vendor is `VENDOR_UNKNOWN or VENDOR_SYNERGENIX`:

```
#define UNKNOWN_UNKNOWN 0
#define UNKNOWN_UNIX    1
#define UNKNOWN_WINDOWS 2
```

Possible values when vendor is `VENDOR_SONYERICSSON:`

```
#define SONYERICSSON_T300 0
```

Video capabilities

```
typedef struct {
   uint16_t size;
   uint16_t flags;
   uint16_t width;
   uint16_t height;
} VIDEOCAPS;
```

```
Members:
```

•`size`	Must be set to `sizeof(VIDEOCAPS)`
•`flags`	Video capability flags
•`width`	Width of screen
•`height`	Height of screen

Video capability flags	Description
VCAPS_3D	Indicates 3D support
VCAPS_GRY2	Monochrome display
VCAPS_GRY4	Four levels of gray
VCAPS_GRY16	Sixteen levels of gray
VCAPS_IND2	Two color indexed display
VCAPS_IND4	Four color indexed display
VCAPS_IND16	Sixteen color indexed display
VCAPS_IND256	256 color indexed display
VCAPS_RGB332	256 color RGB (332) display
VCAPS_16BPP	16-bit (HI-color) RGB display

Input capabilities

```
typedef struct {
   uint16_t size;
   uint16_t flags;
   uint8_t keycount;
} INPUTCAPS;
```

```
Members:
```

•`size`	Must be set to sizeof(INPUTCAPS)
•`flags`	Input capability flags:
•`keycount`	Nr of keys

Input capability flags	Description
ICAPS_POINTER	Platform supports pointer input, for example a pen or mouse
ICAPS_JOYSTICK	Platform has a joystick device
ICAPS_ASCII	Platform supports ASCII text input
ICAPS_NUMERIC_KEYPAD	Platform has numeric keypad as commonly found mobile phones.

Sound capabilities

```
typedef struct {
   uint16_t size;
   uint16_t flags;
   uint8_t  channels;
} SOUNDCAPS;
```

```
Members:
```

- `size` Must be set to sizeof(SOUNDCAPS)
- `flags` Sound capability flags.
- `channels` Maximum number of sound channels supported.

Sound capability flags	Description
SCAPS_BEEP	Platform supports a beeper device.
SCAPS_WAVE	Platform supports wave (PCM) output.
SCAPS_STEREO	Platform supports stereo sound.
SCAPS_MIDI	Platform supports midi music.

Communication capabilities

```
typedef struct {
   uint16_t size;
   uint16_t flags;
} COMMCAPS;
```

```
Members:
```

- `size` Must be set to sizeof(COMMCAPS)
- `flags` Communication capability flags:

Communication capability flags	Description
CCAPS_CABLE	Platform supports serial communications (RS232, USB).
CCAPS_TCP	Platform supports the TCP/IP protocol.
CCAPS_UDP	Platform supports the UDP datagram protocol.
CCAPS_BLUETOOTH	Platform supports bluetooth communication.
CCAPS_IR	Platform supports infra-red communication.
CCAPS_SMS	Platform supports SMS extensions.
CCAPS_FILE	Platform supports file communication.
CCAPS_OBEX	System uses OBEX for BLUETOOTH, IR and CABLE.

2D VIDEO LIBRARY

Header: `#include <vmgp.h>`

int32_t vWaitVBL(int32_t *block*);

- `block` Non-zero if blocking.

Waif for vertical blank (screen refresh). If this function is blocking it waits until vertical blank occurs and then returns TRUE. If this function is non-blocking it returns TRUE if the device is in vertical blank.

Example: Do a pageflip in a vertical blank

```
// Main loop: do AI, draw objects etc.
...
while (vWaitVBL(1));
vFlipScreen(0);
```

void vFlipScreen(int32_t *block*);

- `block` Non-zero to wait for vertical blank (blocking).

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

Example: Do a page flip in a vertical blank (without using vWaitVBL)

```
// Main loop, draw objects
...
vFlipScreen(1);
```

void vSetForeColor(int32_t *color*);

- `color` Set new color to be used.

New color for drawing operations. If bit 31 is set this denotes an RGB555 value, otherwise a palette color index.

Example: Set fore color, indexed or rgb.

```
...
// Palette color index
vSetForeColor(255);

// RGB555
vSetForeColor(vRGB(255,0,0));
...
```

void vSetBackColor(int32_t *color***);**

- `color` Set new background color.

Background color is only used for shadows and outlines when using vTextOut and vTextOutU. The function is used in the same way as vSetForeColor.

int32_t vSetTransferMode(int32_t *mode***);**

- `mode` Transfer mode.

Set the current transfer mode used when drawing objects.

Mode	**Meaning**
MODE_BLOCK	Copy all pixels.
MODE_TRANS	Copy all non-transparent pixels (color 0 is transparent).

Returns: The previous transfer mode is returned.

Example: Set transfer mode to transparent.

```
...
vSetTransferMode(MODE_TRANS);
...
```

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

- `x` The horizontal coordinate where object is drawn.
- `y` The vertical coordinate.
- `object` Pointer to object.

Draws an object at the specified location using the current transfer mode.

Example: Draws an object on the screen.

```
...
SPRITE sprite = { … };
...
vDrawObject(20,30,&sprite);
...
```

```
typedef struct {
   uint8_t     palindex;   // Sprite palette offset
   uint8_t     format;     // Sprite color format
   int16_t     centerx;
   int16_t     centery;
   int16_t     width;
   int16_t     height;
} SPRITE;
```

The format field may be set to one of the following color format specifiers:

Sprite format	Valid 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 …

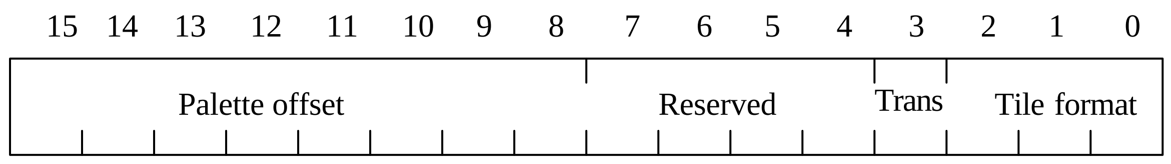
The sprite palette offset is only used for the VCAPS_IND2, VCAPS_IND4 and VCAPS_IND16 formats and the value of palindex is not allowed to exceed 254, 252 and 240 for each format respectively.

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

•`data`	Pointer to tile data.
•`format`	Format of tile.
•`x`	Horizontal position.
•`y`	Vertical position.

Draw a tile of 8x8 pixels in given format at (x, y).

Format:



The following tile formats (bit 0-2) are available:

Format	**Description**
VCAPS_IND2	Two color indexed
VCAPS_IND4	Four color indexed
VCAPS_IND16	Sixteen color indexed
VCAPS_IND256	256 color indexed
VCAPS_RGB332	256 color direct RGB (332)

The transparent bit (nr 3) is 1 for transparent tiles and 0 for solid tiles.

Bits 4-7 are reserved.

The palette offset is located in bit 8-15, and has to be even (0,2,4,6,8…).

Example: Draws a solid tile on the screen.

```
...
uint8_t tiledata[64] = { … };
...
vDrawTile(tiledata, VCAPS_RGB332, 12, 23);
...
```

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

•`x0`	Horizontal position, start of line.
•`y0`	Vertical position, start of line.
•`x1`	Horizontal position, end of line.
•`y1`	Vertical position, end of line.

Draw a line using the current foreground color from (x0, y0) to (x1, y1) inclusive.

Example: Draws a line on the screen.

```
...
vDrawLine(10,10,20,20);
...
```

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

•`x`	Horizontal position.
•`y`	Vertical position.

Draw a pixel using the current foreground color at (x, y).

Example: Plot a pixel on the screen.

```
...
vPlot(12,23);
...
```

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

•`x0`	Horizontal position, left side.
•`y0`	Vertical position, top side.
•`x1`	Horizontal position, right side.
•`y1`	Vertical position, bottom side.

Set the clip window coordinates for all drawing functions. Clip window is inclusive.
If (x0, y0) are set to zero and (x1, y1) are off screen the clip window will be set to the whole screen. Setting an invalid window will result in the clip window remaining unchanged. This occurs if (x0, y0) are off the screen OR if x0>x1 or y0>y1.

vSetClipWindow affects all drawing functions except for vClearScreen.

Example:

```
...
vSetClipWindow(0,0,320,240);
...
```

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

Set the display window used by the 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

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

•`x0`	Horizontal position, left side.
•`y0`	Vertical position, top side.
•`x1`	Horizontal position, right side.
•`y1`	Vertical position, bottom side.

Fill the specified rectangle with the current foreground color. vFillRect supports y0<y1 and/or x0<x1. Coordinates are inclusive.

void vClearScreen(int32_t *color***);**

•`color`	The color to fill the screen with.

Fill the whole screen with a specified color regardless of current foreground color and clip window setting. If bit 31 (in color) is set this denotes an RGB555 value, otherwise a palette color index.

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

•`pal`	Pointer to palette
•`index`	Start entry in palette
•`count`	The number of entries in the palette.

Set the current palette starting at entry index from a pointer to an array of 16 bit color values. The palette uses 5 bits for each color component.

Example: Sets a palette with 16 entries.

```
uint16_t palette[16] = { 0x0000, ... };

...
vSetPalette(palette, 0, sizeof (palette) / 2 );
...
```

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

•`index`	Index to set.
•`rgb`	new RGB value.

Set palette entry `index` to the `rgb` value.

uint32_t vGetPaletteEntry(uint8_t *index***);**

- `index` Index to get.

Returns: The RGB555 color value of a specific palette entry. Sets bit 31.

int32_t vFindRGBIndex(uint32_t *rgb***);**

- `rgb` RGB to find.

Returns a palette entry index that closely matches the `rgb` value.

uint32_t vRGB(uint8_t *r*, uint8_t *g*, uint8_t *b***);**

- `r` Red value.
- `g` Green value.
- `b` Blue value.

Returns: Returns a packed RGB555 value.

uint32_t vCreateGrayValue(uint32_t *rgb***);**

- `rgb` `RGB value to convert.`

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

void vDrawFlatPolygon(VMGPPOLY * v0**);**

- `v0` Pointer to VMGPPOLY.

```
typedef struct
{
  int16_t  x1;
  int16_t  y1;
  int16_t  x2;
  int16_t  y2;
  int16_t  x3;
  int16_t  y3;
} VMGPPOLY;
```

Fill the specified polygon with the current foreground color.

Example: Draws a polygon on the screen.

```
VMGPPOLY v0;

...
v0.x1=x1;
...
vDrawFlatPolygon(&v0);
...
```

VMGPFONT* vSetActiveFont(VMGPFONT **pfont***);**

- `pfont` Pointer to a VMGPFONT structure describing the font.

Set the font used by the vPrint function. The VMGPFONT structure is defined as follows:

```
typedef struct {
    uint8_t *fontdata;  // Font bitmap data
    uint8_t *chartbl;   // Character table
    uint8_t bpp;
    uint8_t width;      // Width of a character
    uint8_t height;     // Height of a character
    uint8_t palindex;   // Palette offset for non-monochrome fonts
} VMGPFONT;
```

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. As of mophun version 1.0 bit depths of 1 and 2 bits per pixel are supported for fonts.

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.

Returns: The previously selected font.

void vPrint(int32_t *mode*, int32_t *x*, int32_t *y*, char **ptr***);**

- `mode` Transfer mode
- `x` Horizontal position.
- `y` Vertical position.
- `ptr` Pointer to string

Display a NULL terminated string on the screen using font previous initialized by vSetActiveFont.

Mode	**Meaning**
MODE_BLOCK	Copy all pixels.
MODE_TRANS	Copy all non-transparent pixels (color 0 is transparent).

Example: Prints a string on the screen.

```
...
vPrint(MODE_TRANS,30,40,”HELLO MOPHUNWORLD”);
```

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

- size — Font size. Can be one of the following:
  - FONT_SIZE_NORMAL
  - FONT_SIZE_SMALL
  - FONT_SIZE_LARGE

- `flags` — Font flags. Can be one or more of the following flags:
  - FONT_STYLE_NORMAL
  - FONT_STYLE_ITALIC
  - FONT_STYLE_BOLD
  - FONT_STYLE_UNDERLINE
  - FONT_STYLE_MONOSPACE

  One of the following effects may also be specified:
  - FONT_EFFECT_OUTLINE
  - FONT_EFFECT_SHADOW_LOWERRIGHT
  - FONT_EFFECT_SHADOW_LOWERLEFT
  - FONT_EFFECT_SHADOW_UPPERRIGHT
  - FONT_EFFECT_SHADOW_UPPERLEFT

- ch — A character that is part of the character set needed. May be zero.

Select the font that is used by subsequent calls to the functions: vTextOut, vTextExtent and vCharExtent. The current fore color is used for the font and the current back color is used for the shadow or outline (see example).

Returns: The font flags used or –1 on error.

Example: Prints a string on the screen.

```
...
vClearScreen(1);

vSetForeColor(255);  // FONT COLOR
vSetBackColor(123);  // OUTLINED/SHADOW COLOR

vSelectFont(FONT_SIZE_SMALL, FONT_STYLE_NORMAL, '0');
vTextOut(10,10,"mophun");

vSelectFont(FONT_SIZE_LARGE, FONT_STYLE_BOLD |
            FONT_EFFECT_SHADOW_LOWERRIGHT, '0');
vTextOut(10,25,"mophun");

vFlipScreen(1);
...
```

Note: On Sony Ericsson terminals the size set by vSelectFont is dependant on the display font size on the terminal.
FONT_SIZE_NORMAL = the current display font size on the terminal.
FONT_SIZE_SMALL = one font size smaller than current display font size on the terminal (never smaller than small).
FONT_SIZE_LARGE = one font size bigger than current display font size on the terminal. (never bigger than big).

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

- x Horizontal position.
- y Vertical position.
- str String that shall be printed

Display a NULL terminated string on the screen using the currently selected font. Note that this is using system fonts rather than the font set by vSetActiveFont.

void vTextOutU(int16_t *x*, int16_t *y*, const wchar_t **str***);**

- x Horizontal position.
- y Vertical position.
- str String that shall be printed

Display a NULL terminated 16-bit UNICODE string on the screen. See vTextOut for details.

int32_t vTextExtent(const char **str***);**

- str NULL-terminated string to measure.

Measure the horizontal and vertical extents of a NULL-terminated string.

Returns: The height is returned in the upper 16 bits, the width in the lower sixteen bits. On error –1 is returned.

int32_t vTextExtentU(const wchar_t **str***);**

- str NULL-terminated string to measure.

Measure the horizontal and vertical extents of a NULL-terminated 16-bit UNICODE string.

Returns: The height is returned in the upper 16 bits, the width in the lower sixteen bits. On error –1 is returned.

int32_t vCharExtent(uint16_t *ch***);**

- ch Character to measure.

Measure the horizontal and vertical extents of a character. The character may be a Unicode character.

Returns: The height is returned in the upper 16 bits, the width in the lower sixteen bits. On error –1 is returned.

2D BACKGROUND MAPPING LIBRARY

Header: `#include <vmgp.h>`

The 2d Mapping functions are used for drawing whole block mapped area's, such as scrolling or static backgrounds.

uint32_t vSpriteInit(uint8_t *count* **);**

•`count`	Number of sprite slots to use

Creates and initialize sprite slots.

Returns Non-zero if successful, zero if request failed.

void vSpriteDispose(void**);**

Dispose sprite engine.

void vSpriteSet(uint8_t *slot*, SPRITE **sprite*, int16_t *x*, int16_t *y*<b>);</b>

•`slot`	Slot number to use.
•`sprite`	Pointer to sprite object.
•`x`	Horizontal position.
•`y`	Vertical position.

Insert sprite into selected slot

void vSpriteClear(void**);**

Clear all sprite slots.

int16_t vSpriteCollision(uint8_t *slot*, uint8_t *slotfrom*, uint8_t *slotto*<b>);</b>

•`slot`	Slot of main sprite to check collision against.
•`slotfrom`	Slot start of sprite(s) to check collision against.
•`slotto`	Slot end of sprite(s) to check collision against.

Check collision between sprites.
Returns the slot number for the first sprite that collides with the provided slot index.
If no collision is detected –1 is returned.
If *slotto* is out of bounds (more then count-1 set by **vSpriteInit**) –1 will be returned.

int16_t vSpriteBoxCollision(VMGPRECT *box, uint8_t *slotfrom*, uint8_t *slotto***);**

•`box`	box to check collision against.
•`slotfrom`	Slot start of sprite(s) to check collision against.
•`slotto`	Slot end of sprite(s) to check collision against.

Check collision between sprite(s) and a box.
Returns : the slot number for the first sprite that collides with the box.
If no collision is detected –1 is returned.
If *slotto* is out of bounds (more then count-1 set by **vSpriteInit**) –1 will be returned.

```
typedef struct
{
  int16_t  x;
  int16_t  y;
  uint16_t width;
  uint16_t height;
} VMGPRECT;
```

uint32_t vMapInit(MAP_HEADER *map**);**

•`map`	Pointer to a mapheader structure.

Initialize map functions with the settings specified in `map`.

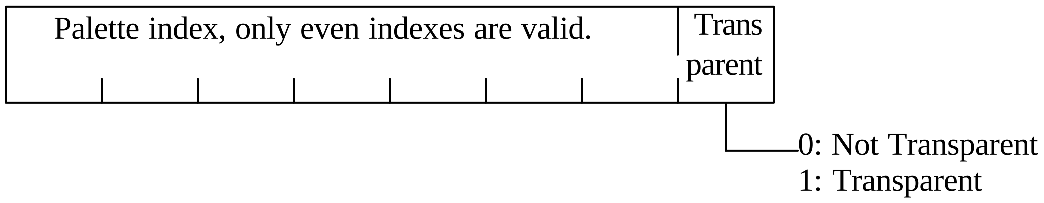
Returns: Non-zero if successful, zero if request failed.

```
typedef struct {
   uint8_t     flag
   uint8_t     format
   uint8_t     width
   uint8_t     height
   uint16_t    xpan
   uint16_t    ypan
   uint16_t    x
   uint16_t    y
   uint8_t     animationspeed
   uint8_t     animationcount       // For internal use only
   uint8_t     animationactive      // For internal use only
   uint8_t     *mapoffset
   uint8_t     *tiledata
} MAP_HEADER;
```

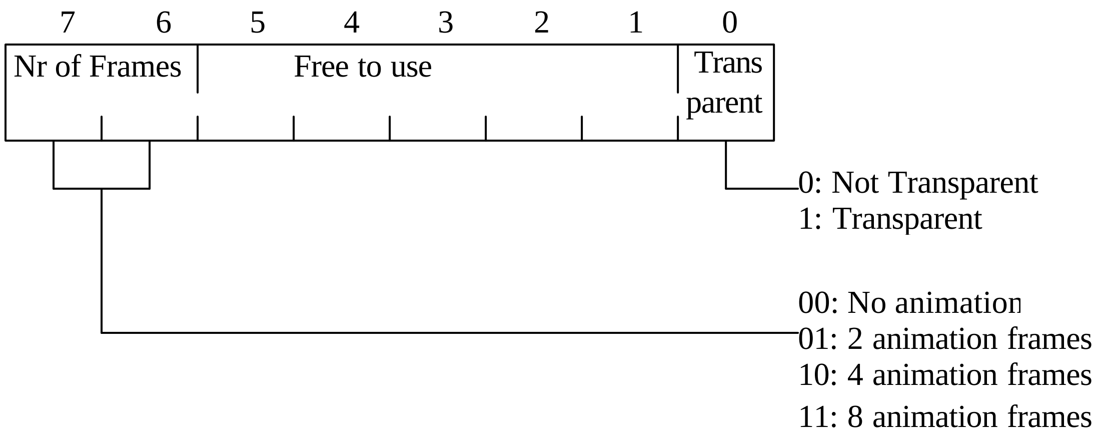
Member	Description
Flag	•MAP_USERATTRIBUTE The tile map have an attribute layer. •MAP_TRANSPARENT The tile map have transparent tiles in the layer. •MAP_AUTOANIM The tile map have auto animated tiles in the layer. One or more flags may be combined.
Format	•VCAPS_IND2 Monochrome index tilemap. •VCAPS_IND4 Four index color tilemap. •VCAPS_IND16 16 Index color tilemap. •VCAPS_IND256 256 Index color tilemap. •VCAPS_RGB332 RGB332 Direct color mode.
Width	Number of tiles in horizontal direction.
Height	Number of tiles in vertical direction.
xpan	Number of pixel to pan map window in horizontal direction relative to screen window.
ypan	Number of pixel to pan map window in vertical direction relative to screen window.
x	Horizontal pixel position relative to mapstart.
y	Vertical pixel position relative to mapstart.
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.
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.
tiledata	Pointer to linear tile data.

Attribute format

If tile format is VCAPS_ IND2, VCAPS_ IND4 or VCAPS_IND16:



If tile format is VCAPS_RGB332 or VCAPS_IND256:



Example: Renders a map.

```
extern unsigned char*tileset;
extern unsigned char*mapdata;
...
void renderBackground(uint16_t xscreen, uint16_t yscreen)
{
   MAP_HEADER MapHeader;

   // HEADER
   MapHeader.flag=0;
   MapHeader.xpan=0;
   MapHeader.ypan=0;
   MapHeader.width     = 16;
   MapHeader.height    = 16;
   MapHeader.mapoffset = mapdata;
   MapHeader.tiledata  = tileset;
   MapHeader.format    = VCAPS_RGB332;

   vMapInit(&MapHeader);
   vMapSetXY(xscreen,yscreen);
   vUpdateMap();
}
```

void vMapDispose(void**);**

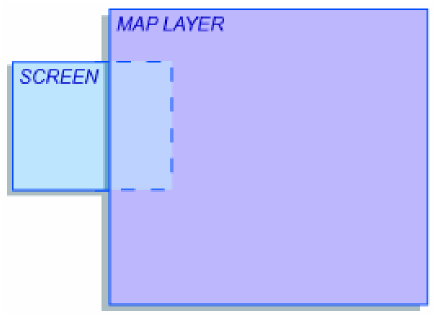
Dispose map engine.

void vUpdateSpriteMap(void);

Draw the current map and sprites.

Note: Coordinates provided for vSetClipWindow is very important when it comes to drawing the background maps. If the clipwindow 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.

Examples:



Map layer in the picture above starts at x-position 60

Example 1: (background may NOT be drawn)
```
vSetClipWindow(0,0,100,80);
vUpdateSpriteMap();
vFlipScreen(1);
```

Example 2: (background will be drawn)
```
vSetClipWindow(60,0,100,80);
vUpdateSpriteMap();
vFlipScreen(1);
```

void vUpdateMap (void);

Draw only the current map.
Note: See vUpdateSpriteMap for instructions regarding clip window settings.

void vUpdateSprite (void);

Draw only the sprites previously initialized by vSpriteInit.

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

- x Horizontal pixel position relative to mapstart.
- y Vertical pixel position relative to mapstart.

Change x/y coordinate offset, used for scrolling.

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

•`x`	Horizontal tile position relative to mapstart.
•`y`	Vertical tile position relative to mapstart.
•`tile`	Tile value.

Change a background tile on the map.

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

•`x`	Horizontal tile position relative to mapstart.
•`y`	Vertical tile position relative to mapstart.
•`attribute`	Attribute value.

Change background attribute in map.

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

•`x`	Horizontal tile position relative to mapstart.
•`y`	Vertical tile position relative to mapstart.

Get tile in map.

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

•`x`	Horizontal tile position relative to mapstart.
•`y`	Vertical tile position relative to mapstart.

Get attribute in map.

uint32_t vMapHeaderUpdate (MAP_HEADER **map***);**

•`map`	Pointer to a mapheader structure.

Updates the mapheader structure.
Returns Non-zero if successful, zero if request failed.

STREAM I/O

Header: `#include <vstream.h>`

The streams I/O system implements a stream I/O interface to communication facilities and file systems.

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

- `name` Specifies the resource to open a stream interface to.
- `mode` Indicates the type and mode of operation of a stream.

Stream Modes	Description
STREAM_FILE	Open a file specified by name
STREAM_CABLE	Open a connection via serial or USB cable
STREAM_TCP	Open a TCP connection to the host specified by name and the IP port specified by the upper 16 bits of the mode parameter
STREAM_UDP	
STREAM_BLUETOOTH	Open a bluetooth connection
STREAM_IR	Open a irDA connection
STREAM_SMS	Send an SMS to the recipient specified by name
STREAM_RESOURCE	Open a resource stream

Mode Flags	Description
STREAM_READ	Open stream for reading
STREAM_WRITE	Open stream for writing. A file is truncated if it exists, otherwise it is created
STREAM_READWRITE	Open stream for reading and writing. If file does not exist it is created
STREAM_CREATE	Create file if it does not exist
STREAM_EXCL	Fail to create file if it already exists
STREAM_TRUNC	Truncate file if it exists and is writeable
STREAM_BINARY	Open file in binary mode. This disable end-of-line handling on some systems
STREAM_OBEX	Open a communications resource in OBEX mode. The initialization and setup of the stream is manufacturer dependent.

The name parameter is interpreted differently depending on the type of stream that is created.

When reading SMS streams, the size of the buffer should be large enough to hold a single SMS, repeated read operations return one SMS message at the time.

Returns: handle to the stream, -1 on error.

void vStreamClose(int32_t handle);

- `handle` The stream handle.

Close the specified stream handle.

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

- `handle` The stream handle.
- `buf` Pointer to a user-supplied buffer.
- `count` The number of bytes to read.

Read bytes from a stream.

Returns: The number of bytes read. –1 on error, 0 indicates an end-of-file condition.

Example: Opens and reads from a file.

```
stream = vStreamOpen("testfil.txt",STREAM_READ);
if(stream != -1)
   {
      res = vStreamRead(stream,str,8);
      vStreamClose(stream);
   }
```

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

- `handle` The stream handle.
- `buf` Pointer to a user-supplied buffer.
- `count` The number of bytes to write.

Write bytes to a stream.

Returns: The number of bytes written. –1 on error.

int32_t vStreamReady(int32_t *handle*, uint32_t mode**);**

- `handle` The stream handle.
- `mode` `STREAM_READ or STREAM_WRITE`.

Checks if reading or writing is completed

Returns: `0` if completed, otherwise `STREAM_READ or STREAM_WRITE (depending on mode)`

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

- `handle` The stream in which to seek.
- `where` The seek offset.
- `whence` The position that the offset is relative to.
  May be: VSEEK_SET, VSEEK_END or VSEEK_CUR.

Seek within a stream. Not all stream types support this operation, currently only STREAM_FILE and STREAM_RESOURCE support seeking. Seeking in files not opened in binary mode may give unexpected results.

Returns: The position within the stream.

int32_t vStreamMode(int32_t *handle***);**

- `handle` The stream handle.

Returns: The current mode of an open stream.

int32_t vResOpen(int32_t *handle*, uint16_t *entry***);**

- `handle` A module handle that indicates the module containing resources. 0 = main module.
- `entry` The resource entry to open.

Open a resource for reading.

Returns: A handle to the stream, -1 on error.

int32_t vResOpenMode(int32_t *handle*, uint16_t *entry*, int32_t *mode***);**

- `handle` A module handle that indicates the module containing resources. 0 = main module.
- `entry` The resource entry to open.
- `mode` Stream mode flags.

Open a resource for in a specific mode.

Returns: A handle to the stream, -1 on error.

void vResClose(int32_t *handle***);**

- `handle` The stream handle.

Close the specified resource handle.

int32_t vResRead(int32_t *handle*, void **buf*, inte32_t *count***);**
See vStreamRead.

int32_t vResWrite(int32_t *handle*, const void **buf*, int32_t *count***);**
See vStreamWrite.

unsigned long ntohl(unsigned long *netlong***);**

- `netlong` A 32-bit value in network byte order (big-endian).

Returns: The value in host byte order.

unsigned long htonl(unsigned long *hostlong***);**

- `hostlong` A 32-bit value in host byte order.

Returns: The value in network byte order (big-endian).

unsigned short ntohs(unsigned long *netshort***);**

- `netshort` A 15-bit value in network byte order (big-endian).

Returns: The value in host byte order.

unsigned short htons(unsigned long *hostshort***);**

- `hostshort` A 16-bit value in host byte order.

Returns: The value in network byte order (big-endian).

SYSTEM LIBRARY

Header: `#include <vmgp.h>`

void *memcpy(void **dst*, const void **src*, uint32_t *n***);**
Copy a block of bytes.

void *memmove(void **dst*, const void **src*, uint32_t *n***);**
Copy a block of bytes and handle overlapping data.

void *memset(void **dst*, uint8_t *val*, uint32_t *size***);**
Set a block of bytes to specific value.

uint32_t vMemFree(void**);**
Returns the number of bytes available on the heap.

uint32_t vMaxFreeBlock(void**);**
Returns the largest available block of bytes available on the heap.

void* vNewPtr(uint32_t *size***);**
Allocate memory from the heap.
Returns: Pointer to the allocated block, NULL on failure.

void vDisposePtr(void **ptr***);**
Deallocate memory previously allocated with `NewPtr`.

uint32_t vFrameTickCount(void**);**
Return the frame count since program start.

uint32_t vGetTickCount(void**);**
Return the current millisecond tick-counter.

void vTerminateVMGP(void**);**
Terminate the application.

uint32_t vGetVMGPInfo(void**);**
Return the VMGP version with major version encoded in high nibble, minor version in low nibble.

void vGetTimeDate(TIMEDATE **tm***);**
Return current time and date.

```
typedef struct {
   uint16_t year;
   uint8_t  month;
   uint16_t day;
   uint8_t  hour;
   uint8_t  second;
   uint8_t  minute;
} TIMEDATE;
```

uint32_t vGetRandom(void**);**
Get a pseudo-random number. `VRAND_MAX` is a constant that defines the largest value that may be returned from this function.

Example: Get a random number from 1 to 6.

```
diceValue = (vGetRandom()%6) + 1;  // Return a number (1-6)
```

void vSetRandom(uint32_t seed);
Set the seed value for the random number generator.

uint32_t vUID(void**);**
Get unique identifier for host terminal. This returns a hashed and encrypted version of the device IMEI (International Manufacturer Equipment Identity) number.

int vCheckIMEI(const char **imei***);**
- `imei` pointer to a 14 character string containing an IMEI number without dashes.

Returns: Non-zero if the supplied IMEI number matches that of the mobile device.

int vCheckDataCert (const void **datacert***);**
- `datacert` Pointer to a data certificate followed by data.

Returns: Non-zero if the certificate is valid and the data has not been tampered with.

int32_t vStrLen(const char **str***);**
Return length of a zero-terminated string.

char *vStrCpy(char **s1*, const char **s2***);**
Copy the zero-terminated string s2 to s1.

Returns: Pointer to the terminating ‘\0’, or *s1* if an error occurs.

char *vitoa(int32_t *val*, char **buf*, uint8_t *len*, uint8_t *pad***);**
Convert an integer to a string and copy it to *buf*. The resulting string is padded to *len* characters of value *pad*.

Returns: Pointer to the terminating '\0', or *buf* if an error occurs.

char *vutoa(uint32_t *val*, char **buf*, uint8_t *len*, uint8_t *pad***);**
Like vitoa but value is unsigned.

Returns: Pointer to the terminating '\0', or *buf* if an error occurs.

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

Display a message box and wait for user input. If neither VMB_YESNO nor VMB_OKCANCEL is set, a simple OK message box is displayed. The contents of the screen are undefined after vMsgBox returns.

Flag	Description
VMB_YESNO	Allow user to select yes or no.
VMB_OKCANCEL	Allow user to select ok or cancel.
VMB_SMALL	Show a small message box. This is the default.
VMB_BIG	Show a large message box.
VMB_ERROR	Message box displays an error message.
VMB_WARNING	Message box displays a warning message.
VMB_INFO	Message box displays an informational message.
VMB_QUESTION	Message box displays question to the user.
VMB_TITLE	If set vMsgBox takes a third parameter that specifies a string to be used as the title of the message box.

Returns: The users choice: VMB_OK, VMB_CANCEL, VMB_NO or VMB_ YES. –1 on error.

int32_t vMsgBoxU(int32_t *flags*, const wchar **msg*, . . .**);**
Like vMsgBox except all strings are UNICODE.

uint16_t vSwap16(uint16_t *u16***);**
Swap byte-order of a 16-bit value. This function does not do anything on platforms that have VMGP native byte-order (little-endian).

uint32_t vSwap32(uint32_t *u32***);**
Swap byte-order of a 32-bit value. This function does not do anything on platforms that have VMGP native byte-order (little-endian).

void vSwap(void **buf*, uint32_t *n*, uint32_t *size***);**
Swap byte-order of all values in a buffer. The number of elements is specified by n, and the size of the elements (in bytes) is specified by size.

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

- `cmd` The system control code determine the action performed by vSysCtl.
- `opt` Command option.

System-control extension.

Possible system control codes are:	**Description**
`SYSCTL_SETVIBRATE`	Control device vibrator. The high 16 bits specify the time in mille-seconds during which vibrator is turned on. Bit 0 determine whether vibrator is turned on.
`SYSCTL_VENDOR`	Not a control code. Vendor specific control codes must use SYSCTL_VENDOR as base. For example: `#define SYSCTL_FLASH (SYSCTL_VENDOR+1)`

void vSetVibrate(on, time**);**

Turn vibrator off or on for a number of milliseconds. Equivalent to:

```
vSysCtl(SYSCTL_SETVIBRATE, on | (time << 16))
```

INPUT LIBRARY

Header: `#include <vmgp.h>`

uint32_t vGetPointerPos(void);
Returns the current position of the pointer. The Y position is encoded in bits 16-31, X is encoded in bits 0-15.

uint32_t vGetButtonData(void);
Returns the VMGP buttons encoded as a bit field. Possible values for the buttons are:

- KEY_UP
- KEY_DOWN
- KEY_LEFT
- KEY_RIGHT
- KEY_FIRE
- KEY_FIRE2
- KEY_SELECT
- POINTER_DOWN
- POINTER_ALTDOWN

Bits 16-31 are reserved for future expansion.

Example: Gets state of the buttons.

```
keys = vGetButtonData();
if(keys&KEY_LEFT)
{
}
```

int32_t vTestKey(uint8_t *ascii*);
Test if the specified ASCII key is pressed.

int32_t vScanKeys(void);
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 bit SCAN_PROCESSING is set, keys are currently being processed and the current key is returned in the low 8 bits. All possible bit settings are list below.

- SCAN_PROCESSING
- SCAN_CTRL
- SCAN_ALT
- SCAN_SHIFT
- SCANKEY_MASK

SOUND LIBRARY

Header: `#include <vmgp.h>`

Vmgp has two types of sound system, firstly, a simple beeper system for low-end devices, and secondly a wave based sound system for higher-end PDA and devices. The beeper system is able to play a single beep for x duration or a sequence of beeps from a resource.

The wave-based sound is able to operate with one or more channels of mono wave sound.
The devices sound system is broken down into x channels (described in the capabilities interface), every even channel is left, and every odd channel is right, on stereo systems, otherwise all of the channels are mono.

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

•`freq`	The frequency in hertz.
•duration	The length of the beep in milli-seconds.

Play a beep on the beep device. vBeep(0,0) will mute the beeper. The beep is asynchronous.

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

•`data`	Pointer to the resource data or a stream handle.
•`length`	Length of the resource in bytes.
•`flags`	Playback flags and resource type.

The vPlayResource plays back a resource or sound asynchronously. Currently the function supports the following data types:

• SOUND_TYPE_BEEP	A sequence of beep frequencies and durations.
• SOUND_TYPE_MIDI	A SMF midi tune.

The least significant bits of flags specify the type of the resource. In addition to the type the following flags may be specified:

`SOUND_FLAG_LOOP`	Loop the sound until a new sound is started or the sound is stopped.
`SOUND_FLAG_STREAM`	If specified the data parameter is a stream handle from which the data is fetched. Only file and resource streams are safe to use. The stream is closed by the system and if it is used the results are undefined.
`SOUND_FLAG_STOP`	Stop the currently playing sound of the specified type.

Returns: Non-zero on success.

Example: Play sounds.

```
int midifd = vResOpen (NULL, BKGTUNE);
vPlayResource ((void*)midifd, BKGTUNE_SIZE,
               SOUND_TYPE_MIDI | SOUND_FLAG_STREAM | SOUND_FLAG_LOOP);
...
vPlayResource (&die, sizeof (die), SOUND_TYPE_BEEP);
```

TASK LIBRARY

Header: `#include <vmgp.h>`

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

Creates a new task that will start execution at `TaskAddr`, which should be the address of a function that accepts up to three parameters (specified in `p0-p2`).

Example: Create a second task.

```
void mytask (SPRITE *spr)
{
   for (;;)
   {
      // Move tank
      ...
      vDrawObject (x, y, spr);
      vSleep(); // Yield to next task
      if (vReceive () != 0)
         vKillTask (); // I’m dead
   }
}

main ()
{
   // Create another task
   int tid = vCreateTask (mytask, &player, 0, 0);
   for (;;)
   {
      vSleep (); // Let other tasks run
      ...
      vFlipScreen (1);
   }
}
```

void vDisposeTask(int32_t *Task***);**

Dispose a task.

Int32_t vTaskAlive(int32_t *Task***);**

Return true if a task is valid.

void vSleep(void**);**

Put current task to sleep and proceed to the next task (if any).

void vKillTask(void**);**

Kill the current task and proceed to the next task (if any). If the current task is the main task the program is terminated.

int32_t vThisTask(void**);**
Return the handle of the current task.

int32_t vReceive(void**);**
Read data from the task data channel. Other tasks can send data across the data channel by calling vSend().

int32_t vReceiveAny(int32_t *Task***);**
Like vReceive() except any task can be specified.

void vSend(int32_t *Task*, int32_t *value***);**
Send data to a specific task.

void vYieldToSystem(void**);**
Put the currently running task to sleep and give control to the system.

uint32_t vSetStackSize(uin32_t *size***);**

- `size` The stack size in bytes.

Set the size of the stack for subsequently created tasks.

Returns: The actual size of the task stack. The value may differ from the value specified because of rounding.

COMPRESSION LIBRARY

Header: `#include <vmgp.h>`

int32_t vDecompHdr(COMPRESSEDFILE *info, uint8_t *src);

- `*info` Pointer to destination. **COMPRESSEDFILE** structure provided by user. This can be set to **NULL** if only the decompressed file size is wanted.
- `*src` Pointer to the compressed data in memory.

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

Returns: The decompressed data size. –1 if error

```
typedef struct
{
   uint8_t     cnt;           // nr countbits used
   uint8_t     offset;        // sliding windows size
   uint16_t    crc16;         // crc16 checksum (of uncompressed data)
   uint16_t    version;       // version flag
   uint16_t    option;        // option flag
   uint32_t    srcsize;       // compressed size
   uint32_t    dstsize;       // uncompressed size
   uint32_t    literalsize;   // reserved for future use
} COMPRESSEDFILE;
```

int32_t vDecompress(uint8_t *src, uint8_t *dst, int32_t handle, uint32_t ReadBuffSize);

- `*src` Pointer to compressed data in memory. If decompressing from stream be sure to set this pointer to **NULL**.
- `*dst` Pointer to destination.
- handle Stream handle from a stream opened by the user. This parameter is ignored if the **src** pointer is not **NULL**.
- ReadBufferSize Readbuffersize used when reading from stream. **vDecompress** manages the allocation and deallocation of this memory. ReadBuffSize must be set to at least 80 bytes (more then 1KB is recommended). ReadBuffSize is ignored if **src** is not **NULL.**

Decompress a file located in memory or from a stream.
Selection between stream or memory decompression is done by setting the src to NULL for stream or a valid pointer for memory decompression.

Returns: If Successful decompression the function returns the decompressed file size. Otherwise –1 is returned for error.

Example: Decompresses from resource.

```
res = vResOpen(NULL,PICTURE1_IN_RESOURCE);
if(res > -1)
{
   vDecompress(NULL,outputmemory,res,1024);
   vResClose(res);
}
```

DATA CERTIFICATE LIBRARY

Header: `#include <vmgp.h>`

Synergenix links in a library used for data certificate checks at certification time. The game filesize will increase by 1500 bytes. The functions in the library can be found in this chapter.

Data Certificate Format

The data certificate always contains the data signature and a default header, the optional data then follows if the data certificate contains any. The data signature is used to verify that the data in the certificate has not been tampered with.

Data Signature(200 bytes)
Header(28 bytes)
[tags](1-256 bytes)
[data](0-8 bytes)
Padding(16 bytes)

Picture1. A Data Certificate

The header in the data certificate contains the following structure:

```
typedef struct{
  uint8_t   IdType;           // The type of id can be IMEI number, UID or none
  uint8_t   IMEI[15];         // IMEI number
  uint32_t  UID;              // Unique ID (used if no IMEI number)
  uint32_t  GameID;           // Game ID
  uint32_t  CertID;           // Certificate ID
} CertInfo;
```

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
```

int16_t vlDataCertCheck(uint8_t *cert**);**

- *cert Pointer must point to a certificate in memory.

- Checks that the data certificate is valid.
- Checks that the game ID in the data certificate matches the game ID inside the game. (the game ID is inserted into the game by Synergenix at certification time)
- If the DRM is activated it also checks to see if it contains the correct IMEI or UID (a unique ID based on the IMEI) so that it is bought for this exact phone.
- If timebomb dates are supplied in tags it checks to see if the game is started within the specified dates (only the first time the game is started).
- If a subscription period is supplied in tags it checks that the game is started in that period.

Returns:

- DATACERT_SUCCESS if none of the checks above fail
- DATACERT_FAILURE if any of the checks for data certificate, game ID or DRM fails.
- DATACERT_EXPIRED if the timebomb or subscription check fails

int32_t vlDataCertGetTagDataSize(uint8_t *cert**);**

- *cert Pointer must point to a certificate in memory.

Returns: the size of the tag and data section.

int32_t vlDataCertGetTagStart(void);

The tags section of the certificate is reserved for dynamic data the distributor adds. The tags are separated by a ":". For example sending in

```
-tag lives=5:TTL=60:
```

to the VST/certtest yields the following in the tags section of the data certificate:

```
lives=5:TTL=60:
```

Even if your game does not require tags there may still be tags inserted by the distributor to handle dates for a subscription based model for example. All tags starting with "M" (as in mophun) are reserved for this, see above.

The tags section ends with a \0 character and is followed by a data section. The data section of the certificate is from an external file that is added to the data certificate, these files have to be available to the distributor in order to be inserted at signing time. The file could contain new credits, levels etc.

Returns: the start position for the tag section in any certificate.

int32_t vlDataCertGetCertID(uint8_t *cert);

- *cert Pointer must point to a certificate in memory.

Returns: the certificate ID if success, otherwise DATACERT_FAILURE. The certificate ID can be used by the 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.

API VERSION

Link with -mversion tag to set the api version used.
Example: -mversion=1.30

Differences between versions

Version 1.30:

- Animationspeed = 0 now animates every frame (animated every second frame in previous versions).
- TileMaps doesn't have to fill whole ClipWindow rect 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.

APPENDIX A: META DATA

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

Below is a list of standard meta data tags.

KEY	**EXPLANATION**	**LENGTH (unicode characters)**	**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 “about box”.
Copyright info	A copyright string	60	Displayed in an “about box”.
Vendor	The name of the vendor	60	Displayed in an “about box”.
ForwardLock	Used when beaming games to protect the revenues when the payment model is pay-per-download.	4	When a game arrives on a phone the ForwardLock is checked and if it is "Yes" the copyright protection is set in the file system, effectively disabling forwarding.
Program version	The version number of the game	6	Displayed in an “about box”. Must be in the format MAJOR.MINOR

All fields are null terminated UNICODE strings. The length includes the terminating ‘\0’ sign.

INDEX