

# HEL Library Reference Manual

Generated by Doxygen 1.4.4

Sun Oct 9 20:20:55 2005



# Contents

<b>1</b>	<b>HEL Library Documentation</b>	<b>1</b>
<b>2</b>	<b>HEL Library Module Index</b>	<b>3</b>
2.1	HEL Library Modules . . . . .	3
<b>3</b>	<b>HEL Library Page Index</b>	<b>5</b>
3.1	HEL Library Related Pages . . . . .	5
<b>4</b>	<b>HEL Library Module Documentation</b>	<b>7</b>
4.1	Debug Functions/Information . . . . .	7
4.2	Background Functions . . . . .	14
4.3	Text Functions . . . . .	16
4.4	Bitmap Mode Functions . . . . .	20
4.5	DMA Functions . . . . .	22
4.6	Fade Functions . . . . .	25
4.7	Interrupt Functions . . . . .	27
4.8	Map Functions . . . . .	31
4.9	Math Functions . . . . .	51
4.10	Object Functions . . . . .	52
4.11	Object Rotation/Scaling Functions . . . . .	68
4.12	Pad Functions . . . . .	71
4.13	Palette Functions . . . . .	74
4.14	SRAM Functions . . . . .	84
4.15	Splash-Screen Functions . . . . .	85
4.16	BIOS Functions . . . . .	87
4.17	System Control Functions . . . . .	89

4.18	<a href="#">Tile Functions</a>	90
4.19	<a href="#">Timer Functions</a>	96
4.20	<a href="#">Window Functions</a>	100
<b>5</b>	<b>HEL Library Page Documentation</b>	<b>107</b>
5.1	<a href="#">Improving compile time</a>	107
5.2	<a href="#">Contact</a>	108
5.3	<a href="#">Donate</a>	109
5.4	<a href="#">FAQ - Frequently Asked Questions</a>	110
5.5	<a href="#">Installation</a>	112
5.6	<a href="#">Introduction</a>	113
5.7	<a href="#">License</a>	114
5.8	<a href="#">Featured Projects</a>	115
5.9	<a href="#">Version History</a>	121

# Chapter 1

## HEL Library Documentation

### Version:

1.7

### Related Pages:

- [Introduction](#)
- [License](#)
- [Featured Projects](#)
- [FAQ - Frequently Asked Questions](#)
- [Donate](#)
- [Contact](#)
- [Installation](#)
- [Improving compile time](#)
- [Version History](#)

### HEL function reference:

- [Background Functions](#)
- [BIOS Functions](#)
- [Bitmap Mode Functions](#)
- [DMA Functions](#)
- [Fade Functions](#)
- [Interrupt Functions](#)
- [Map Functions](#)

- [Math Functions](#)
- [Object Functions](#)
- [Object Rotation/Scaling Functions](#)
- [Pad Functions](#)
- [Palette Functions](#)
- [Splash-Screen Functions](#)
- [SRAM Functions](#)
- [System Control Functions](#)
- [Text Functions](#)
- [Tile Functions](#)
- [Timer Functions](#) (not completely integrated yet)
- [Window Functions](#)

**HEL debugfunction reference:**

- [Debug Functions/Information](#)
  - [Assertion Checking](#)
    - \* [On Assertion](#)
  - [Debug Message Output](#)
  - [Program execution breakpoints](#)
  - [Debug and Release Library mystery](#)

## Chapter 2

# HEL Library Module Index

### 2.1 HEL Library Modules

Here is a list of all modules:

Debug Functions/Information . . . . .	7
Background Functions . . . . .	14
Text Functions . . . . .	16
Bitmap Mode Functions . . . . .	20
DMA Functions . . . . .	22
Fade Functions . . . . .	25
Interrupt Functions . . . . .	27
Map Functions . . . . .	31
Math Functions . . . . .	51
Object Functions . . . . .	52
Object Rotation/Scaling Functions . . . . .	68
Pad Functions . . . . .	71
Palette Functions . . . . .	74
SRAM Functions . . . . .	84
Splash-Screen Functions . . . . .	85
BIOS Functions . . . . .	87
System Control Functions . . . . .	89
Tile Functions . . . . .	90
Timer Functions . . . . .	96
Window Functions . . . . .	100





## Chapter 3

# HEL Library Page Index

### 3.1 HEL Library Related Pages

Here is a list of all related documentation pages:

Improving compile time . . . . .	<a href="#">107</a>
Contact . . . . .	<a href="#">108</a>
Donate . . . . .	<a href="#">109</a>
FAQ - Frequently Asked Questions . . . . .	<a href="#">110</a>
Installation . . . . .	<a href="#">112</a>
Introduction . . . . .	<a href="#">113</a>
License . . . . .	<a href="#">114</a>
Featured Projects . . . . .	<a href="#">115</a>
Version History . . . . .	<a href="#">121</a>



## Chapter 4

# HEL Library Module Documentation

### 4.1 Debug Functions/Information

#### Pages:

- [Assertion Checking](#)
  - [On Assertion](#)
- [Debug Message Output](#)
- [Program execution breakpoints](#)
- [Debug and Release Library mystery](#)

#### 4.1.0.1 Assertion Checking

The `HEL_ASSERT` mechanism evaluates an expression and, when the result is `FALSE`, displays an errorscreen and aborts the program execution until you press a specific button. The error-screen displays several informations where in the source the assertion failed. This includes:

- Filename
- Line
- Function
- Expression
- Description



Asse

### Macro:

```
HEL_ASSERT(expression, message ...)
```

### Parameter:

```
expression    : Expression that evaluates to TRUE or FALSE
message       : Message to display when expression is FALSE
```

The HEL\_ASSERT macro is typically used to identify logic errors during program development by implementing the expression argument to evaluate to FALSE only when the program is operating incorrectly.

After debugging is complete, assertion checking can be turned off by undefine HAM\_DEBUGGER in 'ham/include/mygba.h':

```
#define HAM_DEBUGGER
```

### Example:

In this program the SetHeroState function uses HEL\_ASSERT to test if the passed pointer pHero is NULL and State is lesser than HERO\_STATE\_ITEMS.

```
typedef struct _THero
{
    int State;
}THero;

enum
{
    HERO_STATE_JUMP=0,
    HERO_STATE_WALK,
    HERO_STATE_STAY,

    HERO_STATE_ITEMS
}HERO_STATE;

THero gHero;
```

```
void SetHeroState(THero *pHero, u32 State);

void main(int argc, char *argv[])
{
    // Initialization and whatever ...

    // Does not raise an error!
    SetHeroState(&gHero, HERO_STATE_STAY);

    // Displays an error, because we
    // pass NULL as first parameter!!
    SetHeroState(NULL, HERO_STATE_STAY);
}

void SetHeroState(THero *pHero, u32 State)
{
    HEL_ASSERT(pHero != NULL, "pHero must not be NULL");
    HEL_ASSERT(State < HERO_STATE_ITEMS, "State must not be greater than HERO_STATE_ITEMS");

    pHero->State = State;
}
```

**See also:**

[On Assertion](#)

#### 4.1.0.2 Program execution breakpoints

HEL provides an easy to use mechanism to set program execution breakpoints, which are automatically turned off when you switch to releasemode. Please note that these breakpoints are currently only supported by NO\$GBA emulator and must be explicit turned on!

Using such a breakpoint is easy, all you need is `HEL_DEBUG_BRK`

##### **Example:**

The following example shows how to stop the program execution using `HEL_DEBUG_BRK` macro:

```
int main()
{
    ham_Init();

    // stop program execution!
    HEL_DEBUG_BRK;

    while(GameIsRunning)
    {
        // some fancy stuff ...
    }

    return 0;
}
```

The program execution is stopped after `ham_Init` and before the while-loop. NO\$GBA halts at this point and displays the assembler sourcecode in its debugger-window where you then can step through:

08000230	E24CB004	sub	r11,r12,#0x4	
08000234	E1A0E00F	mov	r14,r15	
08000238	E12FFF10	bx	r0	
0800023C	E1A0B00B	.brk		
08000240	EAF7FFFE	b	#0x8000240	;←

**Note:**

If you use HEL's breakpoint implementation with another emulator than NO\$GBA, it will just leave the corresponding opcode away, assuming you specified what emulator you use (see [Debug Message Output](#)) and therefore will not halt at the position where the breakpoint is set.

**Requirements:**

NO\$GBA shareware or professional version. "Source Code Breakpoints" must be turned on in no\$gba Xcept setup.

**4.1.0.3 Debug and Release Library mystery**

This document is intended to outline the differences between the debug and release library of HEL.

HEL Library comes in two forms. One for debug purposes, using [Assertion Checking](#) and one for the final release, which does not include ASSERT and is faster and smaller.

The debug library consumes more memory (ROM/IWRAM/EWRAM) than the release library and is also slower and bigger in its filesize. The more memory consumption is achieved because it includes a few variables for reference-counting which are located in EWRAM (< 500byte) and some self-modifying-code which must be in IWRAM (< 400byte).

While working on a project, it's recommended to use the debug library. This is just because the debug library includes ASSERT checking and so it will display an error message when you pass an invalid parameter to one of HEL's functions. ASSERT calls seem to be a bit overhead for the little GBA, but they help a lot in the development process and therefore they are implemented in the debug version of HEL.

All ASSERT calls, Reference-Counter variables as well as Debugging Functions are removed in the release library. Even a lot of functions get replaced with macros when you switch to release mode to gain maximum speed. As of version HEL 1.5, this includes the Object/Window/DMA/System/SpecialFX/etc... modules!

**Switch between Debug and Release Library:**

Since sooner or later you want to switch between the debug and release library, here is an explanation how this can be done.

Copy libhel.a as described in the [Installation](#) guide into the correct directory. Now, depending on what version you copied (debug or release) you have to either enable or disable HAM\_DEBUGGER which is located in mygba.h. The file mygba.h is the header file of HAM and located into the 'ham/include' directory in your HAM installation.

Make a stringsearch in this file for HAM\_DEBUGGER. You should come across this text:

```
#define HAM_DEBUGGER 1
```

That is the default setting! If you want to turn off debugging just comment out this line as demonstrated below:

```
//#define HAM_DEBUGGER 1
```

HEL is now also forced to use macros for a lot of functions, please see notes in this context.

#### When to use the Release Library:

A disadvantage of the debug library is that you don't have the full performance as mentioned earlier. At some point of time you want to see how your game really runs on the GBA. This is a good moment to rebuild the project in release mode and then watch it in all its glory and full speed on hardware. Basically, whenever I think my game is bugfree and I want to test it on hardware, I rebuild the entire project with the release library and switch back to the debug version when I continue developing by using an emulator.

#### Notes:

Due to the fact that several modules in HEL are almost entirely replaced with macros in release mode, you can encounter an error that it cannot find a symbol (function) in the library. This error occurs when you use the release library but you did not remove the HAM\_DEBUGGER define. In this case HEL still thinks it should use the functions which are located in the library instead of using the macros.

#### See also:

[Installation](#)

#### 4.1.0.4 Debug Message Output

HEL provides an easy to use mechanism to output formatted debug messages, which are automatically turned off when you switch to releasemode.

The Debug-Message-System currently supports NO\$GBA and VisualBoy Advance. Activating one or the other is easily done:

- Open 'ham/include/hel.h'
- Search for HEL\_DEBUG\_MSG\_TYPE\_ (should be located around line 60)
- Define either HEL\_DEBUG\_MSG\_TYPE\_VBA for VisualBoy Advance Debug-Messages, or HEL\_DEBUG\_MSG\_TYPE\_NOHASH for NO\$GBA support.

HEL\_DEBUG\_MSG\_TYPE\_NOHASH is the default setting when you just installed HEL.

**Example:**

The following example shows how to switch to VisualBoy Advance support. Once you have hel.h in front of you and scrolled a few lines down, you see:

```
//#define HEL_DEBUG_MSG_TYPE_VBA
#define HEL_DEBUG_MSG_TYPE_NOHASH
```

As you can see, at this time NO\$GBA support is activated. To activate VBA, change it to:

```
#define HEL_DEBUG_MSG_TYPE_VBA
//#define HEL_DEBUG_MSG_TYPE_NOHASH
```

**Usage:**

To make use of Debug-Messages in your project, all you need is HEL\_DEBUG\_MSG. It does not matter what Debug-System you activated (NO\$GBA or VBA), just use HEL\_DEBUG\_MSG. Everything else is done for you automatically!

**Macro:**

```
HEL_DEBUG_MSG(txt, args...)
```

**Parameter:**

```
txt      : Format-control string
args     : Optional arguments
```

**Example:**

```
HEL_DEBUG_MSG("Hello World\n");
HEL_DEBUG_MSG("I am %d years old.\n", 28);
```

**Output:**

```
Hello World
I am 28 years old.
```

**Remarks:**

HEL's Debug-Message-System only works, as the name already says, in Debug-Mode. To switch between Debug and Release-Mode, please see [Debug and Release Library mystery](#). Another important part is to know that VisualBoy Advance Debug-Messages do not work on hardware! They freeze the GBA! Having Debug-Messages enabled when using it outside an emulator makes no sense anyway.

**Requirements:**

NO\$GBA shareware or professional version, freeware version does not support debugmessages! VisualBoy Advance SDL (commandline tool).

**See also:**

[Assertion Checking](#)



#### 4.1.0.5 On Assertion

You can specify a function that gets called before an assertion takes place. This can be helpful if you want to output your engine stats or whatever, when something went wrong. Use the `hel_DebugSetOnAssert` function for this.

**Function:**

```
hel_DebugSetOnAssert(PDebugAssertFunc pFunc)
```

**Parameter:**

`pFunc` : A pointer to the function that should be called before an assertion takes place

**Example:**

```
void main(int argc, char *argv[])
{
    // Initialization and whatever ...

    // set function that gets executed before
    // the assertion takes place
    hel_DebugSetOnAssert(OnAssert);

    // raise an assertion
    ASSERT(0==1, "0 cannot be 1");
}

void OnAssert(TDebugAssertInfo *pInfo)
{
    // just write some text to the console
    HEL_DEBUG_MSG("OnAssert called");
}
```

**See also:**

[Assertion Checking](#)

## 4.2 Background Functions

### Functions

- void [hel\\_BgSetMosaic](#) (u8 BgNo, u8 Value)  
*Set BG Mosaic.*
- void [hel\\_BgSetMosaicSize](#) (u8 HSize, u8 VSize)  
*Set BG Mosaic Size.*
- void [hel\\_BgSetPrio](#) (u8 BgNo, u8 Prio)  
*Set BG Priority.*

### 4.2.1 Function Documentation

#### 4.2.1.1 void [hel\\_BgSetMosaic](#) (u8 *BgNo*, u8 *Value*)

Set BG Mosaic.

The [hel\\_BgSetMosaic](#) function enables or disables mosaic processing for the background specified by BgNo.

#### Parameters:

**BgNo** Background number you want to set mosaic for

**Value** TRUE to enable or FALSE to disable mosaic processing

#### See also:

[hel\\_BgSetMosaicSize](#)

#### 4.2.1.2 void [hel\\_BgSetMosaicSize](#) (u8 *HSize*, u8 *VSize*)

Set BG Mosaic Size.

The [hel\\_BgSetMosaicSize](#) function sets the horizontal and vertical mosaic size for the BG system. The background mosaic size applies to all backgrounds with mosaic enabled ([hel\\_BgSetMosaic](#)).

#### Parameters:

**HSize** Horizontal Size of Mosaic (0..15)

**VSize** Vertical Size of Mosaic (0..15)

#### See also:

[hel\\_BgSetMosaic](#)

**4.2.1.3 void hel\_BgSetPrio (u8 *BgNo*, u8 *Prio*)**

Set BG Priority.

The [hel\\_BgSetPrio](#) function sets the priority of the background specified by *BgNo* to the new value specified by *Prio*.

**Parameters:**

***BgNo*** Background number

***Prio*** The new priority. Must be between 0..3

## 4.3 Text Functions

### Functions

- void [hel\\_CustomTextClear](#) (TCustomTextInfo \*pTextInfo)  
*Clear the entire custom text background.*
- void [hel\\_CustomTextClearRow](#) (TCustomTextInfo \*pTextInfo, u32 y)  
*Clear a row.*
- void [hel\\_CustomTextInit](#) (TCustomTextInfo \*pTextInfo, void \*pSrcTiles, u16 SizeU16, u8 ColMode, u8 PalNo, u8 MapSize, u8 MapRot)  
*Initialize a custom textdraw system.*
- void [hel\\_CustomTextPrint](#) (TCustomTextInfo \*pTextInfo, u32 x, u32 y, char \*pText)  
*Output a string on a tiles mode GBA screen.*
- void [hel\\_CustomTextPrintFmt](#) (TCustomTextInfo \*pTextInfo, u32 x, u32 y, char \*pFmt,...)  
*Output a formatted string on a tiles mode GBA screen.*
- u8 [hel\\_CustomTextType](#) (TCustomTextInfo \*pTextInfo, u32 x, u32 y, u8 Delay, u16 Wait, char \*pText)  
*Output a string on a tiles mode GBA screen letter by letter.*

### 4.3.1 Function Documentation

#### 4.3.1.1 void [hel\\_CustomTextClear](#) (TCustomTextInfo \* *pTextInfo*)

Clear the entire custom text background.

The [hel\\_CustomTextClear](#) function clears the entire custom text background using the space character.

#### Parameters:

*pTextInfo* Pointer to TCustomTextInfo structure

#### Remarks:

If the TextInformation specified by pText Info has no space character set, it will probably display garbage.

#### See also:

[hel\\_CustomTextClearRow\(\)](#)

**4.3.1.2 void hel\_CustomTextClearRow (TCustomTextInfo \* *pTextInfo*, u32 *y*)**

Clear a row.

The [hel\\_CustomTextClearRow](#) function clears the row specified by *y* with the space character.

**Parameters:**

*pTextInfo* Pointer to TCustomTextInfo structure  
*y* The row to clear

**Remarks:**

If the TextInformation specified by *pTextInfo* has no space character set, it will probably display garbage.

**See also:**

[hel\\_CustomTextClear](#)

**4.3.1.3 void hel\_CustomTextInit (TCustomTextInfo \* *pTextInfo*, void \* *pSrcTiles*, u16 *SizeUI6*, u8 *ColMode*, u8 *PalNo*, u8 *MapSize*, u8 *MapRot*)**

Initialize a custom textdraw system.

**Parameters:**

*pTextInfo* Pointer to TCustomTextInfo structure  
*pSrcTiles* Pointer to the tile source data  
*SizeUI6* Size of the tiles to be copied into Tile RAM, in number of 16bit chunks  
*ColMode* Color mode of the tile set: 0 = 16colors/16 palettes 1 = 256 colors / 1 palette  
*PalNo* When *ColMode* is 0, this parameter specifies the palette-bank (0..15)  
*MapSize* The size of the map (0-3). This depends on the *MapRot* parameter! See [ham\\_InitMapEmptySet](#) for more information  
*MapRot* 0 for normal map, 1 for a rotation map

**Note:**

The Custom Text-System only works in tilemode!

**See also:**

[hel\\_CustomTextPrint](#), [hel\\_CustomTextType](#)

**4.3.1.4 void hel\_CustomTextPrint (TCustomTextInfo \* *pTextInfo*, u32 *x*, u32 *y*, char \* *pText*)**

Output a string on a tiles mode GBA screen.

The [hel\\_CustomTextPrint](#) function outputs a string to the GBA display, it does not support formatting.

**Parameters:**

*pTextInfo* Pointer to TCustomTextInfo structure  
*x* X character number the string should start on  
*y* Y character number the string should start on  
*pText* The string to output

**Note:**

This function is faster than [hel\\_CustomTextPrintFmt](#)

**See also:**

[hel\\_CustomTextInit](#), [hel\\_CustomTextPrintFmt](#), [hel\\_CustomTextType](#)

**4.3.1.5 void hel\_CustomTextPrintFmt (TCustomTextInfo \* pTextInfo, u32 x, u32 y, char \* pFmt, ...)**

Output a formatted string on a tiles mode GBA screen.

The [hel\\_CustomTextPrintFmt](#) function outputs a formatted string to the GBA display.

**Parameters:**

*pTextInfo* Pointer to TCustomTextInfo structure  
*x* X character number the string should start on  
*y* Y character number the string should start on  
*pFmt* The Format string (see "printf" documentation in your local C tutorial)

**Note:**

This function is slower than [hel\\_CustomTextPrint](#).

**See also:**

[hel\\_CustomTextInit](#), [hel\\_CustomTextPrintFmt](#), [hel\\_CustomTextType](#)

**4.3.1.6 u8 hel\_CustomTextType (TCustomTextInfo \* pTextInfo, u32 x, u32 y, u8 Delay, u16 Wait, char \* pText)**

Output a string on a tiles mode GBA screen letter by letter.

The [hel\\_CustomTextType](#) function outputs a string to the GBA display letter by letter, like a typing machine.

**Parameters:**

*pTextInfo* Pointer to TCustomTextInfo structure  
*x* X character number the string should start on  
*y* Y character number the string should start on  
*Delay* The Delay of typing. When calling this function once per VBL, it's usually specified in frames.

*Wait* The "time" to Wait after the entire text has been drawn. When calling this function once per VBL, it's usually specified in frames.

*pText* The string to output

**Note:**

This function does not support formatting.

**Returns:**

Returns TRUE when the entire text has been typed and the specified `Wait` value has been reached, otherwise FALSE.

**See also:**

[hel\\_CustomTextInit](#), [hel\\_CustomTextPrintFmt](#), [hel\\_CustomTextType](#)

## 4.4 Bitmap Mode Functions

### Functions

- void [hel\\_BmpClear](#) (u16 Color)  
*Clear screen/backbuffer in mode 3, 4 or 5.*
- u16 \* [hel\\_BmpGetBackBuffer](#) (void)  
*Get a pointer to the backbuffer in mode 4 or 5.*
- void [hel\\_BmpLoad](#) (const void \*pSrc)  
*Load a graphic into VideoRAM.*
- void [hel\\_BmpLoadUnComp](#) (const void \*pSrc, u32 CompressionType)  
*Load a compressed graphic into VideoRAM.*

### 4.4.1 Function Documentation

#### 4.4.1.1 void [hel\\_BmpClear](#) (u16 Color)

Clear screen/backbuffer in mode 3, 4 or 5.

**Parameters:**

*Color* The PaletteIndex in mode4, the Colorvalue in mode 3 and 5

This function clears the screen with the specified color in mode 3, and clears the backbuffer in mode 4 and 5.

**Note:**

In mode 3 and 5, use RGB macro to generate a 15bit BGR colorvalue. This macro can be found in HAM header mygba.h.

#### 4.4.1.2 u16\* [hel\\_BmpGetBackBuffer](#) (void)

Get a pointer to the backbuffer in mode 4 or 5.

**Returns:**

The [hel\\_BmpGetBackBuffer](#) function returns a pointer to the current backbuffer

**Note:**

Only mode 4 and 5 have a backbuffer.



**4.4.1.3 void hel\_BmpLoad (const void \* *pSrc*)**

Load a graphic into VideoRAM.

**Parameters:**

*pSrc* Pointer to source graphic data.

**Note:**

The graphic data has to be in the following format:

- Mode 3 : 240\*160 pixels, 15bit BGR
- Mode 4 : 240\*160 pixels, 8 bit
- Mode 5 : 160\*120 pixels, 15bit BGR

**See also:**

[hel\\_BmpLoadUnComp](#)

**4.4.1.4 void hel\_BmpLoadUnComp (const void \* *pSrc*, u32 *CompressionType*)**

Load a compressed graphic into VideoRAM.

**Parameters:**

*pSrc* Pointer to compressed source graphic data.

*CompressionType* Type of Compression the graphic data uses. This can be one of the COMPRESSION\_TYPE\_\* predefined values.

**Note:**

The graphic data has to be in the following format:

- Mode 3 : 240\*160 pixels, 15bit BGR
- Mode 4 : 240\*160 pixels, 8 bit
- Mode 5 : 160\*120 pixels, 15bit BGR

**See also:**

[hel\\_BmpLoad](#), [hel\\_SwiUnComp](#)

## 4.5 DMA Functions

Description.

### Functions

- void [hel\\_DmaCopy16](#) (void \*pDest, const void \*pSrc, u32 SizeIn16BitWords)  
*Copy memory using DMA Channel 3 in 16BIT mode.*
- void [hel\\_DmaCopy32](#) (void \*pDest, const void \*pSrc, u32 SizeIn32BitWords)  
*Copy memory using DMA Channel 3 in 32BIT mode.*
- void [hel\\_DmaSet16](#) (void \*pDest, u16 SrcValue, u32 SizeIn16BitWords)  
*Fills a block of memory with a specified value using Channel 3 in 16BIT mode.*
- void [hel\\_DmaSet32](#) (void \*pDest, u32 SrcValue, u32 SizeIn32BitWords)  
*Fills a block of memory with a specified value using Channel 3 in 32BIT mode.*
- void [hel\\_DmaSet8](#) (void \*pDest, u8 SrcValue, u32 SizeInBytes)  
*Fills a block of memory with a specified value using DMA Channel 3.*
- void [hel\\_DmaZeroMemory](#) (void \*pDest, u32 SizeInBytes)  
*Fills a block of memory with zeros using DMA Channel 3.*

### 4.5.1 Detailed Description

Description.

DMA (Direct Memory Access) Transfers can be used to copy a block of memory from one location to another and/or to fill a block of memory with a given value. [DMA Functions](#) from this module all use DMA Channel 3!

### 4.5.2 Function Documentation

#### 4.5.2.1 void [hel\\_DmaCopy16](#) (void \* *pDest*, const void \* *pSrc*, u32 *SizeIn16BitWords*)

Copy memory using DMA Channel 3 in 16BIT mode.

The [hel\\_DmaCopy16](#) function copies a block of memory from one location to another using DMA Channel 3 in 16BIT mode.

#### Parameters:

- pDest* Pointer to copy destination  
*pSrc* Pointer to block to copy

*SizeIn16BitWords*,: Size 16BIT Words of block to copy

See also:

[hel\\_DmaCopy32](#)

#### 4.5.2.2 void [hel\\_DmaCopy32](#) (void \* *pDest*, const void \* *pSrc*, u32 *SizeIn32BitWords*)

Copy memory using DMA Channel 3 in 32BIT mode.

The [hel\\_DmaCopy32](#) function copies a block of memory from one location to another using DMA Channel 3 in 32BIT mode.

Parameters:

*pDest* Pointer to copy destination

*pSrc* Pointer to block to copy

*SizeIn32BitWords*,: Size 32BIT Words of block to copy

See also:

[hel\\_DmaCopy16](#)

#### 4.5.2.3 void [hel\\_DmaSet16](#) (void \* *pDest*, u16 *SrcValue*, u32 *SizeIn16BitWords*)

Fills a block of memory with a specified value using Channel 3 in 16BIT mode.

The [hel\\_DmaSet16](#) function fills a block of memory with a specified value using DMA Channel 3 in 16bit mode.

Parameters:

*pDest* Pointer to block to fill

*SrcValue* The 16bit value with which to fill

*SizeIn16BitWords* Size in 16bit words of block to fill

Note:

The destination address specified by *pDest* must be multiply 2.

See also:

[hel\\_DmaSet8](#), [hel\\_DmaSet32](#)

#### 4.5.2.4 void [hel\\_DmaSet32](#) (void \* *pDest*, u32 *SrcValue*, u32 *SizeIn32BitWords*)

Fills a block of memory with a specified value using Channel 3 in 32BIT mode.

The [hel\\_DmaSet32](#) function fills a block of memory with a specified value using DMA Channel 3 in 32bit mode.

**Parameters:**

*pDest* Pointer to block to fill

*SrcValue* The 32bit value with which to fill

*SizeIn32BitWords* Size in 32bit words of block to fill

**Note:**

The destination address specified by *pDest* must be multiply 4.

**See also:**

[hel\\_DmaSet8](#), [hel\\_DmaSet16](#)

**4.5.2.5 void hel\_DmaSet8 (void \*pDest, u8 SrcValue, u32 SizeInBytes)**

Fills a block of memory with a specified value using DMA Channel 3.

The [hel\\_DmaSet8](#) function fills a block of memory with a specified value using DMA Channel 3.

**Parameters:**

*pDest* Pointer to block to fill

*SrcValue* The byte value with which to fill

*SizeInBytes* Size in bytes of block to fill

**Note:**

[hel\\_DmaSet8](#) uses partially DMA Channel 3 in 16Bit mode

**See also:**

[hel\\_DmaSet16](#), [hel\\_DmaSet32](#)

**4.5.2.6 void hel\_DmaZeroMemory (void \*pDest, u32 SizeInBytes)**

Fills a block of memory with zeros using DMA Channel 3.

The [hel\\_DmaZeroMemory](#) function fills a block of memory with zeros using DMA Channel 3.

**Parameters:**

*pDest* Pointer to block to fill with zeros

*SizeInBytes* Size in bytes of block to fill with zeros

**Note:**

[hel\\_DmaZeroMemory](#) uses partially DMA Channel 3 in 16Bit mode.

**See also:**

[hel\\_DmaSet8](#), [hel\\_DmaSet16](#), [hel\\_DmaSet32](#)

## 4.6 Fade Functions

### Functions

- TFadeInfo [hel\\_FadeInit](#) (void \*pPalette, u8 PalMode, u8 ColMode, u8 PalNo)  
*Init a palette fader.*
- u8 [hel\\_FadePalette](#) (TFadeInfo \*pFadeInfo, s32 AmountPerCall)  
*Reset a palette fader structure.*
- void [hel\\_FadeReset](#) (TFadeInfo \*pFadeInfo)  
*Reset a palette fader structure.*

### 4.6.1 Function Documentation

#### 4.6.1.1 TFadeInfo hel\_FadeInit (void \*pPalette, u8 PalMode, u8 ColMode, u8 PalNo)

Init a palette fader.

The [hel\\_FadeInit](#) function initializes a TFadeInfo structure, what you need for later usage with other fading-functions.

#### Parameters:

**pPalette** A pointer to the palette (array) which you want to use to fade either in or out.

**PalMode** The palette mode. 0=background palette, 1=object palette. You can also use one of these defines:

- FADE\_PALETTE\_BG
- FADE\_PALETTE\_OBJ

**ColMode** Color mode. 0=16 colors, 1=256 colors. You can also use one of these defines:

- FADE\_COLORS\_16
- FADE\_COLORS\_256

**PalNo** If you want to fade a 16 color palette, you have to select the palette bank using this parameter. Must between 0..15! If you want to fade a 256 color palette, just set it to 0.

#### Returns:

An initialized TFadeInfo structure

#### See also:

[hel\\_FadePalette](#), [hel\\_FadeReset](#)

#### 4.6.1.2 `u8 hel_FadePalette (TFadeInfo * pFadeInfo, s32 AmountPerCall)`

Reset a palette fader structure.

The [hel\\_FadePalette](#) function fades a palette, depending on the information hold by `pFadeInfo`, either in or out.

**Parameters:**

***pFadeInfo*** Pointer to an initialized `TFadeInfo` structure

***AmountPerCall*** The amount you want to fade the palette. A value of 8 equals to one pixel per go. Use positive values to fade the palette in and negative to fade it out.

**Returns:**

Returns `true` when it completely faded in or out, otherwise `false`

**See also:**

[hel\\_FadeInit](#), [hel\\_FadeReset](#)

#### 4.6.1.3 `void hel_FadeReset (TFadeInfo * pFadeInfo)`

Reset a palette fader structure.

The [hel\\_FadeReset](#) function resets the palette-fader specified by `pFadeInfo`, so you can use it again after it's completely faded in or out.

**Parameters:**

***pFadeInfo*** Pointer to an initialized `TFadeInfo` structure

**See also:**

[hel\\_FadeInit](#), [hel\\_FadePalette](#)

## 4.7 Interrupt Functions

### Functions

- u32 [hel\\_IntrGetType](#) (void \*InterruptProc)  
*Get type of a started interrupt function.*
- u8 [hel\\_IntrIsEnabled](#) (u32 InterruptType)  
*Check if an interrupt is enabled or disabled.*
- void [hel\\_IntrStartHandler](#) (u32 InterruptType, void \*pInterruptHandler)  
*Start an interrupt.*
- void [hel\\_IntrStopAll](#) (void)  
*Stop all interrupts.*
- void [hel\\_IntrStopHandler](#) (u32 InterruptType)  
*Stop an interrupt.*

### 4.7.1 Function Documentation

#### 4.7.1.1 u32 [hel\\_IntrGetType](#) (void \* *InterruptProc*)

Get type of a started interrupt function.

##### Parameters:

***InterruptProc*** Pointer to the interrupt function to get the type from.

##### Returns:

On success, the [hel\\_IntrGetType](#) returns one of these constants:

- INT\_TYPE\_VBL
- INT\_TYPE\_HBL
- INT\_TYPE\_VCNT
- INT\_TYPE\_TIM0
- INT\_TYPE\_TIM1
- INT\_TYPE\_TIM2
- INT\_TYPE\_TIM3
- INT\_TYPE\_SIO
- INT\_TYPE\_DMA0
- INT\_TYPE\_DMA1
- INT\_TYPE\_DMA2
- INT\_TYPE\_DMA3

- INT\_TYPE\_KEY
- INT\_TYPE\_CART

When the function fails it returns 0xffff.

#### 4.7.1.2 u8 hel\_IntrIsEnabled (u32 *InterruptType*)

Check if an interrupt is enabled or disabled.

Use this function to check if the interrupt specified by *InterruptType* is whether enabled or disabled.

##### Parameters:

*InterruptType* Interrupt-Type you want to check. This can be one of the following:

- INT\_TYPE\_VBL
- INT\_TYPE\_HBL
- INT\_TYPE\_VCNT
- INT\_TYPE\_TIM0
- INT\_TYPE\_TIM1
- INT\_TYPE\_TIM2
- INT\_TYPE\_TIM3
- INT\_TYPE\_SIO
- INT\_TYPE\_DMA0
- INT\_TYPE\_DMA1
- INT\_TYPE\_DMA2
- INT\_TYPE\_DMA3
- INT\_TYPE\_KEY
- INT\_TYPE\_CART

##### Returns:

This function returns TRUE when the interrupt specified by *InterruptType* is enabled, otherwise it returns FALSE.

#### 4.7.1.3 void hel\_IntrStartHandler (u32 *InterruptType*, void \* *pInterruptHandler*)

Start an interrupt.

The [hel\\_IntrStartHandler](#) function starts the interrupt specified by *InterruptType*.

##### Parameters:

*InterruptType* The Interrupt-Type you want to start. This can be one of the following:

- INT\_TYPE\_VBL
- INT\_TYPE\_HBL



- INT\_TYPE\_VCNT
- INT\_TYPE\_TIM0
- INT\_TYPE\_TIM1
- INT\_TYPE\_TIM2
- INT\_TYPE\_TIM3
- INT\_TYPE\_SIO
- INT\_TYPE\_DMA0
- INT\_TYPE\_DMA1
- INT\_TYPE\_DMA2
- INT\_TYPE\_DMA3
- INT\_TYPE\_KEY
- INT\_TYPE\_CART

***pInterruptHandler*** An Interrupt-Handler function, it is triggered when an interrupt with `InterruptType` occurs.

See also:

[hel\\_IntrStopHandler](#)

#### 4.7.1.4 void `hel_IntrStopAll` (void)

Stop all interrupts.

This function stops all enabled/started interrupts.

See also:

[hel\\_IntrStopHandler](#)

#### 4.7.1.5 void `hel_IntrStopHandler` (u32 *InterruptType*)

Stop an interrupt.

The [hel\\_IntrStopHandler](#) function stops the interrupt specified by `InterruptType`.

**Parameters:**

***InterruptType*** The Interrupt-Type you want to stop. This can be one of the following:

- INT\_TYPE\_VBL
- INT\_TYPE\_HBL
- INT\_TYPE\_VCNT
- INT\_TYPE\_TIM0
- INT\_TYPE\_TIM1
- INT\_TYPE\_TIM2
- INT\_TYPE\_TIM3
- INT\_TYPE\_SIO

- INT\_TYPE\_DMA0
- INT\_TYPE\_DMA1
- INT\_TYPE\_DMA2
- INT\_TYPE\_DMA3
- INT\_TYPE\_KEY
- INT\_TYPE\_CART

**See also:**

[hel\\_IntrStartHandler](#)

## 4.8 Map Functions

Description.

### Functions

- u32 [hel\\_MapBatchScrollBy](#) (TMapScrollInfo \*pMapInfos, s32 DeltaX, s32 DeltaY, int Count)  
*Scroll multiply layers at the same time.*
- u8 [hel\\_MapBatchScrollByEx](#) (TMapScrollInfo \*pMapInfos, s32 DeltaX, s32 DeltaY, int Count, u32 PrimaryLayer)  
*Scroll multiply layers at the same time.*
- void [hel\\_MapDeInit](#) (TMapScrollInfo \*pMapInfo)  
*De-Initialize a map.*
- void \* [hel\\_MapGetCustomData](#) (TMapScrollInfo \*pMapInfo)  
*Get custom data.*
- TPoint32 [hel\\_MapGetPosition](#) (TMapScrollInfo \*pMapInfo)  
*Get the current map position in tiles.*
- TPoint32 [hel\\_MapGetPositionInPixel](#) (TMapScrollInfo \*pMapInfo)  
*Get the current map position in pixel.*
- TPoint32 [hel\\_MapGetPositionInPixelFrom](#) (TMapScrollInfo \*pMapInfo, u32 TileX, u32 TileY)  
*Get position of a tile.*
- u8 [hel\\_MapGetScrollFlags](#) (TMapScrollInfo \*pMapInfo)  
*Get Scrollflags.*
- u16 [hel\\_MapGetTileAt](#) (TMapScrollInfo \*pMapInfo, u32 ScreenX, u32 ScreenY)  
*Get a maptile.*
- void \* [hel\\_MapGetTilePtrAt](#) (TMapScrollInfo \*pMapInfo, u32 ScreenX, u32 ScreenY)  
*Get a pointer to a maptile.*
- TMapScrollInfo [hel\\_MapInit](#) (u8 BgNo, u16 TilesX, u16 TilesY, u8 Rotation, void \*pMapData)  
*Initialize a map.*

- TMapScrollInfo [hel\\_MapInitEx](#) (u8 BgNo, u16 TilesX, u16 TilesY, u8 Rotation, u8 MapSize, void \*pMapData)  
*Initialize a map.*
- TMapScrollInfo [hel\\_MapInitVirtual](#) (u16 TilesX, u16 TilesY, u8 DataTypeSize, void \*pMapData)  
*Initialize a VirtualMap.*
- u8 [hel\\_MapIsBoundsCheckEnabled](#) (TMapScrollInfo \*pMapInfo)  
*Check if a map uses boundschecking.*
- u8 [hel\\_MapIsParallaxEnabled](#) (TMapScrollInfo \*pMapInfo)  
*Check if parallax is enabled.*
- void [hel\\_MapJumpTo](#) (TMapScrollInfo \*pMapInfo, u32 JumpToFlag)  
*Jump to a border of the map.*
- void [hel\\_MapRedraw](#) (TMapScrollInfo \*pMapInfo)  
*Redraw the map.*
- u8 [hel\\_MapScrollBy](#) (TMapScrollInfo \*pMapInfo, s32 DeltaX, s32 DeltaY)  
*Scroll the map.*
- u8 [hel\\_MapScrollTo](#) (TMapScrollInfo \*pMapInfo, TMapCamScrollInfo \*pMapCamInfo, u32 SpeedX, u32 SpeedY)  
*Scroll map to a specific position automatically.*
- void [hel\\_MapSetBoundsCheck](#) (TMapScrollInfo \*pMapInfo, u8 Value)  
*Setup map bounds checking.*
- void [hel\\_MapSetCallbacks](#) (TMapScrollInfo \*pMapInfo, PMapNotifyFunc pOnTopRowChanged, PMapNotifyFunc pOnLeftColumnChanged)  
*Setup map notify-callbacks.*
- void [hel\\_MapSetCustomData](#) (TMapScrollInfo \*pMapInfo, void \*pCustomData)  
*Set custom data.*
- void [hel\\_MapSetDynamicTileReloading](#) (TMapScrollInfo \*pMapInfo, u32 Enable)  
*Enable/Disable dynamic tile reloading.*
- void [hel\\_MapSetParallax](#) (TMapScrollInfo \*pMapInfo, u8 Value)  
*Enable or disable Parallax support.*
- void [hel\\_MapSetParallaxRatio](#) (TMapScrollInfo \*pMapInfo, FIXED RatioX, FIXED RatioY)

*Set parallax X/Y ratio.*

- void [hel\\_MapSetPosition](#) (TMapScrollInfo \*pMapInfo, u32 X, u32 Y)  
*Set map position in tiles.*
- void [hel\\_MapSetPositionInPixel](#) (TMapScrollInfo \*pMapInfo, u32 X, u32 Y)  
*Set map position in pixels.*
- void [hel\\_MapSetScrollFlags](#) (TMapScrollInfo \*pMapInfo, u32 Flags)  
*Set Scrollflags.*
- void [hel\\_MapTransmitPosition](#) (TMapScrollInfo \*pMapInfo)  
*Transmit position to hardware.*

### 4.8.1 Detailed Description

Description.

HEL's map-system implementation supports large maps. The map-drawer only updates dirty regions, instead of always redrawing the whole map!

The map-drawer is faster than the original HAM map functions. It uses LookUpTables wherever possible to gain maximum speed.

It can be combined with HEL's [Tile Functions](#) to dynamically reload tiles which gives you the possibility to create more extended levels.

It supports parallax scrolling using fixed point math.

It supports Virtual Maps. A Virtual Map in HEL's sense is an invisible datalayer, which can be used as a collision or event map. There is no limit in creating Virtual Map's!

### 4.8.2 Function Documentation

#### 4.8.2.1 u32 [hel\\_MapBatchScrollBy](#) (TMapScrollInfo \*pMapInfos, s32 DeltaX, s32 DeltaY, int Count)

Scroll multiply layers at the same time.

##### Parameters:

**pMapInfos** Pointer to an array of initialized TMapScrollInfo structures

**DeltaX** Amount in pixel to scroll on x axis

**DeltaY** Amount in pixel to scroll on y axis

**Count** The amount of TMapScrollInfo elements in pMapInfo

##### Returns:

FALSE when unable to scroll any layers. Otherwise it returns a flag list with these values:

- 1st BG in MapInfos Array Result:
  - BIT0 - Scrolled at least one pixel on X axis.
  - BIT1 - Scrolled at least one pixel on Y axis.
- 2nd BG in MapInfos Array Result:
  - BIT2 - Scrolled at least one pixel on X axis.
  - BIT3 - Scrolled at least one pixel on Y axis.
- 3rd BG in MapInfos Array Result:
  - BIT4 - Scrolled at least one pixel on X axis.
  - BIT5 - Scrolled at least one pixel on Y axis.
- 4th BG in MapInfos Array Result:
  - BIT6 - Scrolled at least one pixel on X axis.
  - BIT7 - Scrolled at least one pixel on Y axis.

```
// Flags is our flag list result, BgNo is the Background you want to check
// and Axis is either 0 for X, or 1 for Y.
// This assumes you are storing the MapInfos in numerical order in the MapInfos array (0-3)
// If you are not just store the result and AND it against the proper bit as shown above.
// Example: ScrollResult & BIT4 <-- X Axis of 3rd BG stored in MapInfos array.
#define CheckBGScroll(_Flags, _BgNo, _Axis)(_Flags) & (1 << (((_BgNO) << 1) + (_Axis)))

TMapScrollInfo MapInfos[2];

MapInfos[0] = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);
MapInfos[1] = hel_MapInit(1, 4096, 32, FALSE, Layer1_Map);

while(GameLoopActive)
{
    if(NewFrame)
    {
        if(UserPressedRight)
        {
            // Scroll one pixel to the right
            u32 ScrollResult = hel_MapBatchScrollBy(&MapInfos, 1, 0, 2);

            // Check to see if background 0 X axis scrolled
            if(CheckBGScroll(ScrollResult, 0, 0))
            {
                HEL_DEBUG_MSG("BG0 scrolled on X.");
            }

            // Check to see if background 1 X axis scrolled
            if(CheckBGScroll(ScrollResult, 1, 0))
            {
                HEL_DEBUG_MSG("BG1 scrolled on X.");
            }

            // Check to see if BG0 OR BG1 Y axis scrolled
            if(CheckBGScroll(ScrollResult, 0, 1) || CheckBGScroll(ScrollResult, 1, 1))
            {
                HEL_DEBUG_MSG("BG0 or BG1 scrolled on Y.");
            }
        }
    }
}
```

See also:

[hel\\_MapScrollBy](#)

#### 4.8.2.2 u8 hel\_MapBatchScrollByEx (TMapScrollInfo \* *pMapInfos*, s32 *DeltaX*, s32 *DeltaY*, int *Count*, u32 *PrimaryLayer*)

Scroll multiply layers at the same time.

**Parameters:**

*pMapInfos* Pointer to an array of initialized TMapScrollInfo structures

*DeltaX* Amount in pixel to scroll on x axis

*DeltaY* Amount in pixel to scroll on y axis

*Count* The count of TMapScrollInfo elements, passed by *pMapInfo*

*PrimaryLayer* : PrimaryLayer represents the layer-index, in pMapInfos, where you want to get the return value from. The return value is the same as in [hel\\_MapScrollBy](#).

Please read [hel\\_MapScrollBy](#).

See also:

[hel\\_MapBatchScrollBy](#), [hel\\_MapScrollBy](#)

#### 4.8.2.3 void hel\_MapDeInit (TMapScrollInfo \* *pMapInfo*)

De-Initialize a map.

**Parameters:**

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

This function deinitializes a TMapScrollInfo structure as well as the mapset in VRAM.

```
TMapScrollInfo MapInfo;

MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);

// Lots of stuff here ...

hel_MapDeInit(&MapInfo);
```

See also:

[hel\\_MapInit](#)

**4.8.2.4 void\* hel\_MapGetCustomData (TMapScrollInfo \* *pMapInfo*)**

Get custom data.

**Parameters:**

***pMapInfo*** Pointer to an initialized TMapScrollInfo structure

**Returns:**

Returns the custom data what you previously assigned to the structure specified by *pMapInfo*

**See also:**

[hel\\_MapSetCustomData](#)

**4.8.2.5 TPoint32 hel\_MapGetPosition (TMapScrollInfo \* *pMapInfo*)**

Get the current map position in tiles.

**Parameters:**

***pMapInfo*** Pointer to an initialized TMapScrollInfo structure

**Returns:**

Returns the X/Y Coordinates in tiles, as TPoint32 structure

```
TMapScrollInfo MapInfo;
TPoint32 Pt={0,0};

MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);
hel_MapSetPosition(&MapInfo, 512, 0);

Pt = hel_MapGetPosition(&MapInfo);
HEL_DEBUG_MSG("Position in tiles X: %d \n", Pt.X);
HEL_DEBUG_MSG("Position in tiles Y: %d \n", Pt.Y);
```

**See also:**

[hel\\_MapGetPositionInPixel](#)

**4.8.2.6 TPoint32 hel\_MapGetPositionInPixel (TMapScrollInfo \* *pMapInfo*)**

Get the current map position in pixel.

**Parameters:**

***pMapInfo*** Pointer to an initialized TMapScrollInfo structure

**Returns:**

Returns the X/Y Coordinates in pixels, as TPoint32 structure



```

TMapScrollInfo MapInfo;
TPoint32 Pt={0,0};

MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);
hel_MapSetPosition(&MapInfo, 512, 0);

Pt = hel_MapGetPositionInPixel(&MapInfo);
HEL_DEBUG_MSG("Position in pixels X: %d \n", Pt.X);
HEL_DEBUG_MSG("Position in pixels Y: %d \n", Pt.Y);

```

See also:

[hel\\_MapGetPosition](#)

#### 4.8.2.7 TPoint32 hel\_MapGetPositionInPixelFrom (TMapScrollInfo \* pMapInfo, u32 TileX, u32 TileY)

Get position of a tile.

**Parameters:**

**pMapInfo** Pointer to an initialized TMapScrollInfo structure

**TileX** The horizontal index of the tile you want to get the position from

**TileY** The vertical index of the tile you want to get the position from

**Returns:**

Returns the position in pixels of the tile in question.

**Note:**

Imagine the following scenario. You have a layer whose (top/left) position is xy=0,0. Now you want to get the position of the tile at xy=2,3. The result value will be xy=16,24 (2\*8,3\*8). Now lets say you scroll the layer by 2 pixels on x and 6 pixels on y. The result value in this case is xy=14,18 (2\*8-2,3\*8-6). This only works with tiles whose tilesize is 8\*8 pixels.

```
TPoint32 Pt = hel_MapGetPositionInPixelFrom(&g_MapInfo, 2, 3);
```

See also:

[hel\\_MapGetPosition](#), [hel\\_MapGetPositionInPixel](#)

#### 4.8.2.8 u8 hel\_MapGetScrollFlags (TMapScrollInfo \* pMapInfo)

Get Scrollflags.

**Parameters:**

**pMapInfo** Pointer to an initialized TMapScrollInfo structure

**Returns:**

Returns the scrollflags used by the mapsystem specified by pMapInfo. Please see [hel\\_MapSetScrollFlags](#) for a listing of existing flags.

See also:

[hel\\_MapSetScrollFlags](#)

#### 4.8.2.9 **u16** `hel_MapGetTileAt` (TMapScrollInfo \* *pMapInfo*, u32 *ScreenX*, u32 *ScreenY*)

Get a maptile.

**Parameters:**

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*ScreenX* The horizontal screen-coordinate in pixels from where you want the tile from

*ScreenY* The vertical screen-coordinate in pixels from where you want the tile from

**Returns:**

Returns the maptile requested.

**Note:**

This function does not support VirtualMaps, use [hel\\_MapGetTilePtrAt](#) for Virtual-Maps!

```
u16 Index = hel_MapGetTileAt(&g_MapInfo, 98, 56);
```

**See also:**

[hel\\_MapGetTilePtrAt](#)

#### 4.8.2.10 **void\*** `hel_MapGetTilePtrAt` (TMapScrollInfo \* *pMapInfo*, u32 *ScreenX*, u32 *ScreenY*)

Get a pointer to a maptile.

**Parameters:**

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*ScreenX* The horizontal screen-coordinate in pixels from where you want the tile from

*ScreenY* The vertical screen-coordinate in pixels from where you want the tile from

**Returns:**

Returns a pointer to the requested maptile.

**Note:**

You have to cast the pointer to the correct type. For normal maps it is usually an u16, while rotation maps use u8 instead. If you specified a VirtualMap, cast it to the type your VirtualMap items use.

```
u16 Index = *(u16*)hel_MapGetTilePtrAt(&g_MapInfo, 98, 56);
```

**See also:**

[hel\\_MapGetTileAt](#)

#### 4.8.2.11 TMapScrollInfo hel\_MapInit (u8 *BgNo*, u16 *TilesX*, u16 *TilesY*, u8 *Rotation*, void \* *pMapData*)

Initialize a map.

##### Parameters:

***BgNo*** Background you want to initialize the map for  
***TilesX*** Map-width in tiles  
***TilesY*** Map-height in tiles  
***Rotation*** Indicates the map as a rotation map. Can be TRUE or FALSE  
***pMapData*** Pointer to map data

##### Returns:

A TMapScrollInfo structure. This structure is used by almost every hel\_\*Map\* function.

This function returns an initialized TMapScrollInfo structure, what is used by most of the HEL map functions. The hardware map-size is set to 32x32 tiles.

```
TMapScrollInfo MapInfo;

MapInfo = hel_MapInit(0,           // BgNo
                     4096,         // Map-width in tiles
                     32,           // Map-height in tiles
                     FALSE,        // Is it a rotation map?
                     Layer0_Map);  // Pointer to map-data
```

##### See also:

[hel\\_MapInitEx](#), [hel\\_MapDeInit](#)

#### 4.8.2.12 TMapScrollInfo hel\_MapInitEx (u8 *BgNo*, u16 *TilesX*, u16 *TilesY*, u8 *Rotation*, u8 *MapSize*, void \* *pMapData*)

Initialize a map.

##### Parameters:

***BgNo*** Background you want to initialize the map for  
***TilesX*** Map-width in tiles  
***TilesY*** Map-height in tiles  
***Rotation*** Indicates the map as a rotation map. Can be TRUE or FALSE  
***MapSize*** Specifies the hardware map size. See below for a list with available values.  
***pMapData*** Pointer to map data

##### Returns:

A TMapScrollInfo structure. This structure is used by all hel\_Map\* functions.

This function returns an initialized `TMapScrollInfo` structure, which is used by most of the `hel_map` functions. The hardware map is set to 32x32 tiles.

The `MapSize` parameter can be one of the following constants, depending on whether it is a rotation map or not:

For a normal map use one of these:

- `MAP_SIZE_32X32`
- `MAP_SIZE_64X32`
- `MAP_SIZE_32X64`
- `MAP_SIZE_64X64`

For a rotation map use one of these:

- `MAP_SIZE_ROT_16X16`
- `MAP_SIZE_ROT_32X32`
- `MAP_SIZE_ROT_64X64`
- `MAP_SIZE_ROT_128X128`

```
TMapScrollInfo MapInfo;

MapInfo = hel_MapInitEx(0,           // BgNo
                        4096,        // Map-width in tiles
                        512,         // Map-height in tiles
                        TRUE,        // Is it a rotation map?
                        MAP_SIZE_ROT_64X64, // hardware map size
                        Layer0_Map); // Pointer to map-data
```

See also:

[hel\\_MapDeInit](#), [hel\\_MapInit](#)

#### 4.8.2.13 TMapScrollInfo hel\_MapInitVirtual (u16 TilesX, u16 TilesY, u8 DataTypeSize, void \*pMapData)

Initialize a VirtualMap.

##### Parameters:

**TilesX** Map-width in tiles

**TilesY** Map-height in tiles

**DataTypeSize** The size of one element in bytes, from the map specified by *pMapData*

**pMapData** Pointer to mapdata

##### Returns:

A `TMapScrollInfo` structure. This structure is used by almost every `hel_Map*` function.

**Note:**

A VirtualMap in HEL can be any kind of mapdata, but it cannot be displayed! VirtualMap's can be used for collision or event layers for example and must also have a 'tilesize' of 8x8 pixels!

```
TMapScrollInfo MapInfo;  
  
MapInfo = hel_MapInitVirtual(128, 32, sizeof(u16), (void*)sample_Collision);
```

**See also:**

[hel\\_MapInit](#), [hel\\_MapInitEx](#), [hel\\_MapGetTilePtrAt](#)

**4.8.2.14 u8 hel\_MapIsBoundsCheckEnabled (TMapScrollInfo \*pMapInfo)**

Check if a map uses boundschecking.

**Parameters:**

**pMapInfo** Pointer to an initialized TMapScrollInfo structure

**Returns:**

Returns TRUE when boundschecking is enabled, otherwise FALSE.

```
TMapScrollInfo MapInfo;  
  
MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);  
if (hel_MapIsBoundsCheckEnabled(&MapInfo))  
    HEL_DEBUG_MSG("Boundscheck enabled\n");  
else  
    HEL_DEBUG_MSG("Boundscheck disabled\n");
```

**See also:**

[hel\\_MapSetBoundsCheck](#)

**4.8.2.15 u8 hel\_MapIsParallaxEnabled (TMapScrollInfo \*pMapInfo)**

Check if parallax is enabled.

**Parameters:**

**pMapInfo** Pointer to an array of initialized TMapScrollInfo structures

**Returns:**

Returns true when parallax support is enabled in pMapInfo, otherwise false.

**See also:**

[hel\\_MapSetParallax](#)

#### 4.8.2.16 void `hel_MapJumpTo` (TMapScrollInfo \* *pMapInfo*, u32 *JumpToFlag*)

Jump to a border of the map.

The `hel_MapJumpTo` jumps to a border of the map. It can jump to the left or right as well as to the top or bottom. You can combine the `JumpToFlag`'s in order to move the map in more than one direction with only a single call to `hel_MapJumpTo`.

##### Parameters:

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*JumpToFlag* Flag(s) what specify the destination map position. The `JumpToFlag` can be a combination or the following predefined constants:

- `MAP_JUMPTO_LEFT` - Jumps to the very left of the map
- `MAP_JUMPTO_RIGHT` - Jumps to the very right of the map, the last column of the screen (column 30) will display the last column of the map.
- `MAP_JUMPTO_TOP` - Jumps to the very top of the map
- `MAP_JUMPTO_BOTTOM` - Jumps to the very bottom of the map, the last row of the screen (column 20) will display the last row of the map.

Pass zero if you don't want to jump to anything.

##### Remarks:

You cannot jump to the left and right or to the top and bottom simultaneously. Therefore has `MAP_JUMPTO_LEFT` higher priority than `MAP_JUMPTO_RIGHT` and `MAP_JUMPTO_TOP` has higher priority than `MAP_JUMPTO_BOTTOM`. When you pass `MAP_JUMPTO_LEFT` and `MAP_JUMPTO_RIGHT`, only `MAP_JUMPTO_LEFT` will be executed.

```
// Jumps to the very right bottom position
hel_MapJumpTo(&MapInfo, MAP_JUMPTO_RIGHT | MAP_JUMPTO_BOTTOM);
```

##### See also:

[hel\\_MapSetPosition](#)

#### 4.8.2.17 void `hel_MapRedraw` (TMapScrollInfo \* *pMapInfo*)

Redraw the map.

##### Parameters:

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

It's not necessary to call this function directly, though. It's internally used when you set the position using `hel_MapSetPosition` and `hel_MapSetPositionInPixel`. However, it could be handy sometimes, so here it is.

##### Note:

This function is (extremely) slower than `hel_MapScrollBy`

```
TMapScrollInfo MapInfo;  
  
MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);  
  
hel_MapRedraw(&MapInfo);
```

**See also:**

[hel\\_MapScrollBy](#)

**4.8.2.18 u8 hel\_MapScrollBy (TMapScrollInfo \*pMapInfo, s32 DeltaX, s32 DeltaY)**

Scroll the map.

The [hel\\_MapScrollBy](#) function scrolls the map and redraws only the dirty map cells (tiles), instead of redrawing the entire screen.

**Parameters:**

**pMapInfo** Pointer to an initialized TMapScrollInfo structure

**DeltaX** Amount in pixel to scroll on X axis. Positive numbers scroll to the right, negative to the left.

**DeltaY** Amount in pixel to scroll on Y axis. Positive numbers scroll to the bottom, negative to the top.

**Returns:**

Returns FALSE when it was not able to scroll, otherwise it is a combination these values:

- BIT0 - Scrolled at least one pixels on X axis
- BIT1 - Scrolled at least one pixels on Y axis

The return value is always TRUE if the the map does **not** use bounds-checking ([hel\\_MapSetBoundsCheck](#))!

**Note:**

When parallax-support is enabled, the Delta parameters become multiplied by the parallax ratios!

```
TMapScrollInfo MapInfo;  
  
MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);  
  
while(GameLoopActive)  
{  
    if(NewFrame)  
    {  
        if(UserPressedRight)  
        {  
            // Scroll one pixel to the right  
            u8 ScrollResult = hel_MapScrollBy(&MapInfo, 1, 0);  
  
            // check if it was able to scroll on x-axis
```

```

        if(ScrollResult & BIT1)
        {
            HEL_DEBUG_MSG("Scrolled on X");
        }

        // check if it was able to scroll on y-axis
        if(ScrollResult & BIT2)
        {
            HEL_DEBUG_MSG("Scrolled on Y");
        }
    }
}

hel_MapDeInit(&MapInfo);

```

**See also:**

[hel\\_MapSetCallbacks](#), [hel\\_MapSetBoundsCheck](#), [hel\\_MapIsBoundsCheck-Enabled](#), [hel\\_MapSetParallaxRatio](#), [hel\\_MapSetParallax](#)

#### 4.8.2.19 u8 hel\_MapScrollTo (TMapScrollInfo \* *pMapInfo*, TMapCamScrollInfo \* *pMapCamInfo*, u32 *SpeedX*, u32 *SpeedY*)

Scroll map to a specific position automatically.

**Parameters:**

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*pMapCamInfo* Pointer to a TMapCamScrollInfo structure.

*SpeedX* Scroll speed on horizontal axis

*SpeedY* Scroll speed on vertical axis

**Returns:**

- MAP\_CAM\_NEXTKEYPOINT  
Changed to the next keypoint.
- MAP\_CAM\_LASTKEYREACHED  
Scrolling is done. The map has reached the position specified by the very last keypoint in *pMapCamInfo*

Furthermore it returns the value from [hel\\_MapScrollBy\(\)](#), when it was able to scroll. (when none of the above apply)

**See also:**

[hel\\_MapScrollBy](#)

**Note:**

This function scrolls the map automatically to the specified key-points specified in *pMapCamInfo*. The function should be called once per frame.



```

TMapScrollInfo g_MapInfo;

// X/Y Tile-Position (Camera Key-Points)
const TPoint16 CamKeyPoints[5]=
{
    // X, Y Tile Positions
    {10, 5},
    {28, 5},
    {15, 5},
    {15,25},
    { 0, 0}
};

// Map Cinema Scrolling Information
TMapCamScrollInfo g_MapCamInfo =
{
    &CamKeyPoints, // Pointer to keypoints
    5,              // Count of keypoints
    0              // Start keypoint
};

// Init map
g_MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);

u8 CamScrollWorking=1;

while(1)
{
    if(NewFrame)
    {
        if(CamScrollWorking)
        {
            CamScrollWorking = hel_MapScrollTo(&g_MapInfo, &g_MapCamInfo, 1, 1);
        }
        NewFrame=FALSE;
    }
}

```

#### 4.8.2.20 void hel\_MapSetBoundsCheck (TMapScrollInfo \*pMapInfo, u8 Value)

Setup map bounds checking.

##### Parameters:

**pMapInfo** Pointer to an initialized TMapScrollInfo structure

**Value** Boundschecking on/off. Must be 1 or 0

This function enables or disables boundschecking for the structure specified by pMapInfo. This means, you cannot scroll out of the map. It stops scrolling when it hits the "edges" of the map. It's turned on by default when you use [hel\\_MapInit](#).

Disabling this feature means you can scroll out of the map and this could end up in tile-corruption when you do not code an own "is current position still in map" routine.

```
TMapScrollInfo MapInfo;
```

```
MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);
// Turn off boundscheck
hel_MapIsBoundsCheckEnabled(&MapInfo, FALSE);
```

See also:

[hel\\_MapInit](#)

#### 4.8.2.21 void hel\_MapSetCallbacks (TMapScrollInfo \* *pMapInfo*, PMapNotifyFunc *pOnTopRowChanged*, PMapNotifyFunc *pOnLeftColumnChanged*)

Setup map notify-callbacks.

Parameters:

***pMapInfo*** Pointer to an initialized TMapScrollInfo structure

***pOnTopRowChanged*** Pointer to function. It will be triggered when a new row has been drawn.

***pOnLeftColumnChanged*** Pointer to function. It will be triggered when a new column has been drawn.

Note:

Hold the callbacks as small and fast as possible! Otherwise it can happen that the map is updated very weirdly. Think about just setting flags in a callback-function and progress the flags in mainloop, or similar.

```
TMapScrollInfo MapInfo;

// Is called whenever a new row has been drawn
void MapOnTopRowChanged(TMapScrollInfo *pMapInfo, u32 X, u32 Y)
{
    HEL_DEBUG_MSG("MapOnTopRowChanged\n");
    HEL_DEBUG_MSG("X: %d\n", X);
    HEL_DEBUG_MSG("Y: %d\n", Y);
}

// Init map
MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);

// Set callback function(s)
hel_MapSetCallbacks(&MapInfo, (PMapNotifyFunc)MapOnTopRowChanged, NULL);

// Scroll the map by 32 pixels in X and Y
hel_MapScrollBy(&MapInfo, 32, 32);
```

See also:

[hel\\_MapScrollBy](#)

#### 4.8.2.22 void hel\_MapSetCustomData (TMapScrollInfo \* *pMapInfo*, void \* *pCustomData*)

Set custom data.

**Parameters:***pMapInfo* Pointer to an initialized TMapScrollInfo structure*pCustomData* Pointer to the data you want to have assigned to this structure**Note:**

With this function you can assign a pointer, to a custom structure of your game engine for example, to the map-structure specified by `pMapInfo`. This is handy when you work with the HEL map-callback function and need to access additional information.

**See also:**
[hel\\_MapGetCustomData](#), [hel\\_MapSetCallbacks](#)

#### 4.8.2.23 void hel\_MapSetDynamicTileReloading (TMapScrollInfo \* *pMapInfo*, u32 *Enable*)

Enable/Disable dynamic tile reloading.

The [hel\\_MapSetDynamicTileReloading](#) function can be used to force the mapsystem to dynamically reload tiles. Before enabling dynamic-tile-reloading, you must init a tile-system ([hel\\_TileInit](#))!

**Parameters:***pMapInfo* Pointer to an initialized TMapScrollInfo structure*Enable* Set it to TRUE to enable or FALSE to disable

```
// Init the tile-system first
hel_TileInit(0, world_Tiles, 451, BufferA, 224, BufferB, 1, 0);

// Init map
g_MapInfo = hel_MapInit(0, MAP_WORLD_WIDTH, MAP_WORLD_HEIGHT, FALSE, (void*)world_Map);

// Use dynamic tile reloading
hel_MapSetDynamicTileReloading(&g_MapInfo, TRUE);
```

**See also:**
[hel\\_TileInit](#)

#### 4.8.2.24 void hel\_MapSetParallax (TMapScrollInfo \* *pMapInfo*, u8 *Value*)

Enable or disable Parallax support.

**Parameters:***pMapInfo* Pointer to an array of initialized TMapScrollInfo structures*Value* Parallax support

Use this function to enable or disable parallax for the TMapScrollInfo structure specified by `pMapInfo`. Specify TRUE to enable, or FALSE to disable.

**See also:**
[hel\\_MapScrollBy](#), [hel\\_MapIsParallaxEnabled](#)

#### 4.8.2.25 void `hel_MapSetParallaxRatio` (TMapScrollInfo \* *pMapInfo*, FIXED *RatioX*, FIXED *RatioY*)

Set parallax X/Y ratio.

##### Parameters:

*pMapInfo* Pointer to an array of initialized TMapScrollInfo structures

*RatioX* The parallax-ratio on x-axis

*RatioY* The parallax-ratio on y-axis

Use this function to set the parallax ratio for the structure specified by `pMapInfo`. This function automatically enables parallax-support for the specified structure.

```
hel_MapSetParallaxRatio(&MapInfo,
                        PARALLAX_FLOAT_TO_RATIO(0.7), // Ratio on X
                        PARALLAX_FLOAT_TO_RATIO(0.8)); // Ratio on Y
```

##### See also:

[hel\\_MapScrollBy](#), [hel\\_MapIsParallaxEnabled](#)

#### 4.8.2.26 void `hel_MapSetPosition` (TMapScrollInfo \* *pMapInfo*, u32 *X*, u32 *Y*)

Set map position in tiles.

##### Parameters:

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*X* X map-position in tiles

*Y* Y map-position in tiles

This sets the map to the specified position. X and Y must be specified in tiles.

##### Note:

This function sets the left/top position of a map. For example, if you call `hel_MapSetPosition(&MapInfo, 2, 5)`, then is the first column on screen the 2nd column from map and the first row on screen is the 5th row from map.

```
TMapScrollInfo MapInfo;

MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);
hel_MapSetPosition(&MapInfo, 512, 0);
```

##### See also:

[hel\\_MapSetPositionInPixel](#)

#### 4.8.2.27 void `hel_MapSetPositionInPixel` (TMapScrollInfo \* *pMapInfo*, u32 *X*, u32 *Y*)

Set map position in pixels.

##### Parameters:

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*X* X map-position in pixels

*Y* Y map-position in pixels

This sets the map to the specified position. *X* and *Y* must be specified in pixels.

```
TMapScrollInfo MapInfo;

MapInfo = hel_MapInit(0, 4096, 32, FALSE, Layer0_Map);
hel_MapSetPositionInPixel(&MapInfo, 512, 0);
```

##### See also:

[hel\\_MapSetPosition](#)

#### 4.8.2.28 void `hel_MapSetScrollFlags` (TMapScrollInfo \* *pMapInfo*, u32 *Flags*)

Set Scrollflags.

The [hel\\_MapSetScrollFlags](#) function can be used to prevent the mapsystem to scroll into specific direction and to automatically update the map-position in VRAM.

For example: you disable scrolling for the left direction. When you call [hel\\_MapScrollBy](#) with DeltaX lesser than 0 (which indicates you want to scroll to the left), the system will handle the DeltaX parameter as 0 (zero) and therefore will not scroll to the left.

##### Parameters:

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

*Flags* Use these flags to specify in what direction the mapsystem specified by *pMapInfo* can scroll. This can be a combination of these constants:

- MAP\_SCROLLFLAGS\_LEFT - Allows the system to scroll to the left
- MAP\_SCROLLFLAGS\_RIGHT - Allows the system to scroll to the right
- MAP\_SCROLLFLAGS\_UP - Allows the system to scroll upwards
- MAP\_SCROLLFLAGS\_DOWN - Allows the system to scroll downwards
- MAP\_SCROLLFLAGS\_TRANSMITPOSITION - Allows the system to automatically transmit the map position to vram
- MAP\_SCROLLFLAGS\_DEFAULT - Default flags, used when initializing a new map

##### Example:

```
// allow the system to scroll to the left and right as well as
// automatically updating the map position in vram
hel_MapSetScrollFlags(&MapInfo, MAP_SCROLLFLAGS_LEFT | MAP_SCROLLFLAGS_RIGHT | MAP_SCROLLFLAG
```

See also:

[hel\\_MapGetScrollFlags](#), [hel\\_MapScrollBy](#), [hel\\_MapTransmitPosition](#)

#### 4.8.2.29 void [hel\\_MapTransmitPosition](#) (TMapScrollInfo \* *pMapInfo*)

Transmit position to hardware.

The [hel\\_MapTransmitPosition](#) function transmits the position of the map specified by `pMapInfo` to the hardware. Usually it is not necessary to call [hel\\_MapTransmitPosition](#), because it is done automatically when you call [hel\\_MapScrollBy](#) and the ScrollFlags include `MAP_SCROLLFLAGS_TRANSMITPOSITION`. However, if you changed the ScrollFlags and `MAP_SCROLLFLAGS_TRANSMITPOSITION` is not set, you have to transmit the map position with [hel\\_MapTransmitPosition](#).

**Parameters:**

*pMapInfo* Pointer to an initialized TMapScrollInfo structure

See also:

[hel\\_MapSetScrollFlags](#)

## 4.9 Math Functions

### Functions

- void [hel\\_MathExtractDigits](#) (u32 Value, u8 \*pOut, u32 nDigits)  
*Extract digits from a value.*

#### 4.9.1 Function Documentation

##### 4.9.1.1 void [hel\\_MathExtractDigits](#) (u32 Value, u8 \*pOut, u32 nDigits)

Extract digits from a value.

The [hel\\_MathExtractDigits](#) can be used to extract digits from a given value. This is handy when dealing with score display.

#### Parameters:

**Value** Source value from where to extract the digits from

**pOut** Destination buffer which receives the extracted digits

**nDigits** Amount of digits to extract

The following example extracts the digits from 1234. The result is stored in `Buffer` which will contain the following values afterwards: 1,2,3,4

```
u8 Buffer[4];  
  
hel_MathExtractDigits(1234, Buffer, 4);
```

#### Note:

The same result can be achieved with some C modulus/sprintf/whatever. Obviously, this slow! [hel\\_MathExtractDigits](#) is more designed for speed. It uses the BIOS functions instead.

## 4.10 Object Functions

Description.

### Functions

- void [hel\\_ObjBringToFront](#) (u8 ObjNo)  
*Bring an object to front.*
- void [hel\\_ObjClearOAM](#) (void)  
*Clear Object Attribute Memory (OAM).*
- u32 [hel\\_ObjCountVisible](#) (void)  
*Return the number of visible objects.*
- u8 [hel\\_ObjCreate](#) (void \*pSrc, u16 ObjShape, u16 ObjSize, u16 ObjMode, u16 ColMode, u16 PalNo, u16 Mosaic, u16 HFlip, u16 VFlip, u16 Prio, u16 DblSize, s16 X, s16 Y)  
*Create a new object.*
- u8 [hel\\_ObjCreate16](#) (void \*pSrc, u16 ObjShape, u16 ObjSize, u16 ObjMode, u16 PalNo, s16 X, s16 Y)  
*Create a new object using 16 colors.*
- u8 [hel\\_ObjCreate256](#) (void \*pSrc, u16 ObjShape, u16 ObjSize, u16 ObjMode, s16 X, s16 Y)  
*Create a new object using 256 colors.*
- u8 [hel\\_ObjExists](#) (u8 ObjNo)  
*Check if an object exist.*
- u8 [hel\\_ObjGetColorMode](#) (u8 ObjNo)  
*Check if an object uses 256 or 16 colors.*
- u8 [hel\\_ObjGetFirst](#) (void)  
*Get the first object handle.*
- u16 [hel\\_ObjGetGfxSlot](#) (u8 ObjNo)  
*Get the graphic slot of an object.*
- u8 [hel\\_ObjGetMode](#) (u8 ObjNo)  
*Get the mode of an object.*
- u8 [hel\\_ObjGetPrio](#) (u8 ObjNo)  
*Get object priority.*



- u32 [hel\\_ObjGetShape](#) (u8 ObjNo)  
*Get object shape.*
- u32 [hel\\_ObjGetSize](#) (u8 ObjNo)  
*Get object size.*
- u32 [hel\\_ObjGetSizeU16](#) (u8 ObjNo)  
*Get object size in halfwords.*
- void [hel\\_ObjHideAll](#) (void)  
*Hide all objects.*
- u8 [hel\\_ObjIsHFlip](#) (u8 ObjNo)  
*Check if an object is horizontally flipped.*
- u8 [hel\\_ObjIsMosaic](#) (u8 ObjNo)  
*Check if an object has mosaic attribute set.*
- u8 [hel\\_ObjIsRotScale](#) (u8 ObjNo)  
*Check if an object has the rotation/scaling attribute set.*
- u8 [hel\\_ObjIsVFlip](#) (u8 ObjNo)  
*Check if an object is vertically flipped.*
- u8 [hel\\_ObjIsVisible](#) (u8 ObjNo)  
*Check if an object is visible.*
- void [hel\\_ObjSelectRotScale](#) (u8 ObjNo, u8 RotNo)  
*Select a rotation/scaling parameter.*
- void [hel\\_ObjSendToBack](#) (u32 ObjNo)  
*Set an object behind all other objects.*
- void [hel\\_ObjSetHFlip](#) (u8 ObjNo, u8 Value)  
*Set Horizontal Flipping.*
- void [hel\\_ObjSetMode](#) (u8 ObjNo, u8 Mode)  
*Set mode of an object.*
- void [hel\\_ObjSetMosaicSize](#) (u8 HSize, u8 VSize)  
*Set OBJ Mosaic Size.*
- void [hel\\_ObjSetPrio](#) (u8 ObjNo, u32 Prio)  
*Set object priority.*
- void [hel\\_ObjSetRotScale](#) (u8 ObjNo, u8 Value)

*Set a rotation/scaling attribute.*

- void [hel\\_ObjSetVFlip](#) (u8 ObjNo, u8 Value)  
*Set Vertical Flipping.*
- void [hel\\_ObjSetVisible](#) (u8 ObjNo, u8 Value)  
*Set visibility of an object.*
- void [hel\\_ObjSetVisibleAll](#) (u8 Show)  
*Show/Hide all objects.*
- void [hel\\_ObjSetX](#) (u8 ObjNo, s16 X)  
*Set X position.*
- void [hel\\_ObjSetXY](#) (u8 ObjNo, s16 X, s16 Y)  
*Set X and Y position.*
- void [hel\\_ObjSetY](#) (u8 ObjNo, s16 Y)  
*Set Y position.*
- void [hel\\_ObjShowAll](#) (void)  
*Show all objects.*
- u8 [hel\\_ObjToggleVisible](#) (u8 ObjNo)  
*Toggle visibility of an object.*
- void [hel\\_ObjUpdateGfx](#) (u8 ObjNo, void \*pSrc)  
*Update object graphic.*
- void [hel\\_ObjUpdateGfxUnComp](#) (u8 ObjNo, void \*pSrc, u32 Compression-Type)  
*Update object graphic with decompression support.*
- void ATTR\_DEPRECATED [hel\\_ObjUpdateInOAM](#) (u8 ObjNo)  
*Update an object in OAM immediately.*

#### 4.10.1 Detailed Description

Description.

A lot of these [Object Functions](#) are available in HAM too, but the [Object Functions](#) in HEL get replaced with macros when you switch to release-mode. This make them a couple of times faster than the original functions from HAM (HAM 2.8 at this time).

## 4.10.2 Function Documentation

### 4.10.2.1 void `hel_ObjBringToFront` (u8 *ObjNo*)

Bring an object to front.

The [hel\\_ObjBringToFront](#) function brings the object specified by `ObjNo` in front of all other objects.

**Parameters:**

*ObjNo* Object Handle

**See also:**

[hel\\_ObjSendToBack](#)

### 4.10.2.2 void `hel_ObjClearOAM` (void)

Clear Object Attribute Memory (OAM).

The [hel\\_ObjClearOAM](#) function clears the OAM. More precise it sets every OAM entry to a 16Color object with background priority 3, a size of 8x8 pixels which is located at X=241 and Y=161.

**Note:**

This function has no impact on the HAM object management system, it just clears the OAM. All objects and attributes from HAM's shadowed OAM will stay intact.

### 4.10.2.3 u32 `hel_ObjCountVisible` (void)

Return the number of visible objects.

**Returns:**

Returns the number of visible objects in HAM's object management system.

**Note:**

This function does not check if objects are located inside the visible display area, it checks the "isVisible" flag instead ([hel\\_ObjIsVisible](#)).

**See also:**

[hel\\_ObjIsVisible](#), [hel\\_ObjToggleVisible](#), [hel\\_ObjShowAll](#), [hel\\_ObjHideAll](#)

### 4.10.2.4 u8 `hel_ObjCreate` (void \* *pSrc*, u16 *ObjShape*, u16 *ObjSize*, u16 *ObjMode*, u16 *ColMode*, u16 *PalNo*, u16 *Mosaic*, u16 *HFlip*, u16 *VFlip*, u16 *Prio*, u16 *DbISize*, s16 *X*, s16 *Y*)

Create a new object.

[hel\\_ObjCreate](#) creates a new object in HAM.

The [hel\\_ObjCreate](#) is more or less only a wrapper function for `ham_CreateObj`. That's because you cannot specify negative values for either the X or Y coordinate with `ham_CreateObj` (at least for HAM 2.8 or older). The priority parameter seems to be ignored in `ham_CreateObj` as well. Check your local HAM documentation to learn more about the `ham_CreateObj` parameters.

**Returns:**

Handle of the new object.

**See also:**

[hel\\_ObjCreate16](#), [hel\\_ObjCreate256](#)

#### 4.10.2.5 **u8 hel\_ObjCreate16 (void \* *pSrc*, u16 *ObjShape*, u16 *ObjSize*, u16 *ObjMode*, u16 *PalNo*, s16 *X*, s16 *Y*)**

Create a new object using 16 colors.

The [hel\\_ObjCreate16](#) function is a simplified version of [hel\\_ObjCreate](#) for objects with 16 colors. It creates a new object in HAM, using a 16 color palette and it automatically sets these properties:

- `ColMode = 0`
- `Mosaic = FALSE`
- `HFlip = FALSE`
- `VFlip = FALSE`
- `Prio = 0`
- `DblSize = FALSE`

Check your local HAM documentation to learn more about the `ham_CreateObj` parameters.

**Returns:**

Handle of the new object.

**See also:**

[hel\\_ObjCreate](#), [hel\\_ObjCreate256](#)

#### 4.10.2.6 **u8 hel\_ObjCreate256 (void \* *pSrc*, u16 *ObjShape*, u16 *ObjSize*, u16 *ObjMode*, s16 *X*, s16 *Y*)**

Create a new object using 256 colors.

The [hel\\_ObjCreate256](#) function is a simplified version of [hel\\_ObjCreate](#) for objects with 256 colors. It creates a new object in HAM, using a 256 color palette and it automatically sets these properties:

- ColMode = 1
- PalNo = 0
- Mosaic = FALSE
- HFlip = FALSE
- VFlip = FALSE
- Prio = 0
- DbSize = FALSE

Check your local HAM documentation to learn more about `ham_CreateObj` parameters.

**Returns:**

Handle of the new object.

**See also:**

[hel\\_ObjCreate](#), [hel\\_ObjCreate16](#)

**4.10.2.7 u8 hel\_ObjExists (u8 ObjNo)**

Check if an object exist.

The [hel\\_ObjExists](#) function checks if the object specified by `ObjNo` exists in HAM. An object exists when you created it using either the original `ham_CreateObj` function, or one of HEL's wrapper functions: [hel\\_ObjCreate](#), [hel\\_ObjCreate16](#), [hel\\_ObjCreate256](#)

**Parameters:**

*ObjNo* Object Handle

**Returns:**

TRUE if the object exists, otherwise FALSE

**See also:**

[hel\\_ObjIsVisible](#)

**4.10.2.8 u8 hel\_ObjGetColorMode (u8 ObjNo)**

Check if an object uses 256 or 16 colors.

The [hel\\_ObjGetColorMode](#) can be used to check if the object specified by `ObjNo` uses 256 or 16 colors.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

Returns 1 if it uses 256 colors, otherwise 0

**4.10.2.9 u8 hel\_ObjGetFirst (void)**

Get the first object handle.

**Returns:**

Returns the handle (object index) of the first object in HAM.

**4.10.2.10 u16 hel\_ObjGetGfxSlot (u8 *ObjNo*)**

Get the graphic slot of an object.

The [hel\\_ObjGetGfxSlot](#) function returns the graphic slot where the object specified by *ObjNo* points to.

**Returns:**

The graphic slot of the object specified by *ObjNo*. This slot-value is between 0..1023

**4.10.2.11 u8 hel\_ObjGetMode (u8 *ObjNo*)**

Get the mode of an object.

The [hel\\_ObjGetMode](#) function returns what mode an object uses.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

- OBJ\_MODE\_NORMAL
- OBJ\_MODE\_SEMITRSPARENT
- OBJ\_MODE\_OBJWINDOW

**See also:**

[hel\\_ObjSetMode](#)

**4.10.2.12 u8 hel\_ObjGetPrio (u8 *ObjNo*)**

Get object priority.

The [hel\\_ObjGetPrio](#) function returns the priority of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

**See also:**

[hel\\_ObjSetPrio](#)

**4.10.2.13 u32 hel\_ObjGetShape (u8 *ObjNo*)**

Get object shape.

The [hel\\_ObjGetShape](#) function returns the shape-type of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

- OBJ\_SHAPE\_SQUARE
- OBJ\_SHAPE\_HORIZONTAL
- OBJ\_SHAPE\_VERTICAL

**4.10.2.14 u32 hel\_ObjGetSize (u8 *ObjNo*)**

Get object size.

The [hel\\_ObjGetSize](#) function returns the size-type of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

The return value is between 0..3 and forms together with the ObjShape ([hel\\_ObjGetShape](#)) the dimension of the object. See table below:

Size	Square	Horizontal	Vertical
0	8x8	16x8	8x16
1	16x16	32x8	8x32
2	32x32	32x16	16x32
3	64x64	64x32	32x64

**See also:**

[hel\\_ObjGetSize](#), [hel\\_ObjGetShape](#)

**4.10.2.15 u32 hel\_ObjGetSizeU16 (u8 *ObjNo*)**

Get object size in halfwords.

The [hel\\_ObjGetSizeU16](#) function returns the size in halfwords of the graphic where the object specified by *ObjNo* points to.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

Size in halfwords of the graphic where the object specified by `ObjNo` points to. This depends on the `ObjShape` ([hel\\_ObjGetShape](#)), `ObjSize` ([hel\\_ObjGetSize](#)) as well as the `ColorMode` ([hel\\_ObjGetColorMode](#)). An object with an 8bit 16\*32pixel graphic uses 256 halfwords, whereas a 4bit graphic with the same dimension only uses the half amount (128 halfwords). It is computed as followed:

- 8BIT:  $(\text{ObjWidth} * \text{ObjHeight}) / 2$
- 4BIT:  $(\text{ObjWidth} * \text{ObjHeight}) / 4$

**See also:**

[hel\\_ObjGetSize](#), [hel\\_ObjGetShape](#)

**4.10.2.16 void hel\_ObjHideAll (void)**

Hide all objects.

The [hel\\_ObjHideAll](#) function hides all available objects in HAM's object management system.

**See also:**

[hel\\_ObjShowAll](#), [hel\\_ObjSetVisibleAll](#)

**4.10.2.17 u8 hel\_ObjIsHFlip (u8 ObjNo)**

Check if an object is horizontally flipped.

The [hel\\_ObjIsVFlip](#) function checks if the object specified by `ObjNo` is horizontally flipped, or to be more precise, if the horizontal flip bit is set.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

Returns TRUE if it is horizontally flipped, otherwise FALSE

**See also:**

[hel\\_ObjIsVFlip](#)

**4.10.2.18 u8 hel\_ObjIsMosaic (u8 ObjNo)**

Check if an object has mosaic attribute set.

The [hel\\_ObjIsMosaic](#) function can be used to check if the object specified by `ObjNo` has the Mosaic attribute set.

**Parameters:**

*ObjNo* Object Handle



**Returns:**

Returns TRUE if it uses mosaic, otherwise FALSE

**4.10.2.19 u8 hel\_ObjIsRotScale (u8 *ObjNo*)**

Check if an object has the rotation/scaling attribute set.

The [hel\\_ObjIsRotScale](#) function can be used to check if the object specified by *ObjNo* has the *Rotation/Scaling* attribute set.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

TRUE if rotation/scaling is used, otherwise FALSE

**4.10.2.20 u8 hel\_ObjIsVFlip (u8 *ObjNo*)**

Check if an object is vertically flipped.

The [hel\\_ObjIsVFlip](#) function checks if the object specified by *ObjNo* is vertically flipped, or to be more precise, if the vertical flip bit is set.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

Returns TRUE if it is vertically flipped, otherwise FALSE

**See also:**

[hel\\_ObjIsHFlip](#)

**4.10.2.21 u8 hel\_ObjIsVisible (u8 *ObjNo*)**

Check if an object is visible.

The [hel\\_ObjIsVisible](#) function checks if the object specified by *ObjNo* is visible.

**Parameters:**

*ObjNo* Object Handle

**Returns:**

Returns TRUE if it is visible, otherwise FALSE

**See also:**

[hel\\_ObjSetVisibleAll](#), [hel\\_ObjToggleVisible](#)

**4.10.2.22 void hel\_ObjSelectRotScale (u8 *ObjNo*, u8 *RotNo*)**

Select a rotation/scaling parameter.

The [hel\\_ObjSelectRotScale](#) function selects a rotation/scaling parameter of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*RotNo* Rotation/scaling parameter Handle

**See also:**

[hel\\_ObjSetRotScale](#)

**4.10.2.23 void hel\_ObjSendToBack (u32 *ObjNo*)**

Set an object behind all other objects.

The [hel\\_ObjSendToBack](#) function sends the object specified by *ObjNo* behind all other objects.

**Parameters:**

*ObjNo* Object Handle

**See also:**

[hel\\_ObjBringToFront](#)

**4.10.2.24 void hel\_ObjSetHFlip (u8 *ObjNo*, u8 *Value*)**

Set Horizontal Flipping.

The [hel\\_ObjSetHFlip](#) function horizontally flips the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*Value* TRUE to flip, otherwise FALSE

**See also:**

[hel\\_ObjSetVFlip](#)

**4.10.2.25 void hel\_ObjSetMode (u8 *ObjNo*, u8 *Mode*)**

Set mode of an object.

The [hel\\_ObjSetMode](#) function sets the mode of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*Mode* The mode this object has to be set to. This can be one of the following predefined values:

- OBJ\_MODE\_NORMAL
- OBJ\_MODE\_OBJWINDOW
- OBJ\_MODE\_SEMITRSPARENT

**See also:**

[hel\\_ObjGetMode](#)

**4.10.2.26 void hel\_ObjSetMosaicSize (u8 HSize, u8 VSize) [inline]**

Set OBJ Mosaic Size.

The [hel\\_ObjSetMosaicSize](#) function sets the horizontal and vertical mosaic size for the OBJ system. The OBJ mosaic size applies to all objects with mosaic activated ([ham\\_SetObjMosaic](#)).

**Parameters:**

*HSize* Horizontal Size of Mosaic (0..15)

*VSize* Vertical Size of Mosaic (0..15)

**See also:**

[hel\\_ObjCreate](#), [hel\\_ObjIsVisible](#)

**4.10.2.27 void hel\_ObjSetPrio (u8 ObjNo, u32 Prio)**

Set object priority.

The [hel\\_ObjSetPrio](#) function sets the priority of the object specified by *ObjNo*. The object priority is relative to backgrounds.

**Parameters:**

*ObjNo* Object Handle

*Prio* The priority value must be between 0..3! It represents the background number where this object is placed before.

**See also:**

[hel\\_ObjGetPrio](#)

**4.10.2.28 void hel\_ObjSetRotScale (u8 *ObjNo*, u8 *Value*)**

Set a rotation/scaling attribute.

The [hel\\_ObjSetRotScale](#) function can be used to enable or disable rotation/scaling processing for the object specified by `ObjNo`.

**Parameters:**

*ObjNo* Object Handle

*Value* Either TRUE to enable, or FALSE to disable rotation/scaling processing

**See also:**

[hel\\_ObjSelectRotScale](#)

**4.10.2.29 void hel\_ObjSetVFlip (u8 *ObjNo*, u8 *Value*)**

Set Vertical Flipping.

The [hel\\_ObjSetHFlip](#) function vertically flips the object specified by `ObjNo`.

**Parameters:**

*ObjNo* Object Handle

*Value* TRUE to flip, otherwise FALSE

**See also:**

[hel\\_ObjSetHFlip](#)

**4.10.2.30 void hel\_ObjSetVisible (u8 *ObjNo*, u8 *Value*)**

Set visibility of an object.

The [hel\\_ObjSetVisible](#) function either shows or hides the object with the object number specified by `ObjNo`

**Parameters:**

*ObjNo* Object Handle

*Value* TRUE to show, FALSE to hide object

**See also:**

[hel\\_ObjIsVisible](#), [hel\\_ObjSetVisibleAll](#), [hel\\_ObjToggleVisible](#), [hel\\_ObjShowAll](#)

**4.10.2.31 void hel\_ObjSetVisibleAll (u8 *Show*)**

Show/Hide all objects.

The [hel\\_ObjSetVisibleAll](#) function either shows or hides all available objects in HAM's object management system.

**Parameters:**

*Show* Must be TRUE to show or FALSE to hide all objects

**See also:**

[hel\\_ObjHideAll](#), [hel\\_ObjShowAll](#)

**4.10.2.32 void hel\_ObjSetX (u8 *ObjNo*, s16 *X*)**

Set X position.

The [hel\\_ObjSetX](#) function sets the horizontal position of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*X* The screen position, specified in pixels, where the object has to be set to

**See also:**

[hel\\_ObjSetY](#), [hel\\_ObjSetXY](#)

**4.10.2.33 void hel\_ObjSetXY (u8 *ObjNo*, s16 *X*, s16 *Y*)**

Set X and Y position.

The [hel\\_ObjSetXY](#) function sets the horizontal and vertical position of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*X* The screen position, specified in pixels, where the object has to be set to

*Y* The screen position, specified in pixels, where the object has to be set to

**See also:**

[hel\\_ObjSetX](#), [hel\\_ObjSetY](#)

**4.10.2.34 void hel\_ObjSetY (u8 *ObjNo*, s16 *Y*)**

Set Y position.

The [hel\\_ObjSetY](#) function sets the vertical position of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*Y* The screen position, specified in pixels, where the object has to be set to

**See also:**

[hel\\_ObjSetY](#), [hel\\_ObjSetXY](#)

**4.10.2.35 void hel\_ObjShowAll (void)**

Show all objects.

The [hel\\_ObjShowAll](#) function shows all created objects in HAM's object management system.

**See also:**

[hel\\_ObjHideAll](#), [hel\\_ObjSetVisibleAll](#)

**4.10.2.36 u8 hel\_ObjToggleVisible (u8 *ObjNo*)**

Toggle visibility of an object.

The [hel\\_ObjToggleVisible](#) function toggles the visibility of the object with the object number specified by *ObjNo*

**Parameters:**

*ObjNo* Object Handle

**Returns:**

Returns TRUE if the object is visible, otherwise FALSE

**See also:**

[hel\\_ObjIsMosaic](#)

**4.10.2.37 void hel\_ObjUpdateGfx (u8 *ObjNo*, void \* *pSrc*)**

Update object graphic.

The [hel\\_ObjUpdateGfx](#) function updates the graphic of the object specified by *ObjNo*.

**Parameters:**

*ObjNo* Object Handle

*pSrc* Pointer to source graphic

**See also:**

[hel\\_ObjUpdateGfxUnComp](#)

**4.10.2.38 void hel\_ObjUpdateGfxUnComp (u8 *ObjNo*, void \* *pSrc*, u32 *CompressionType*)**

Update object graphic with decompression support.

The [hel\\_ObjUpdateGfxUnComp](#) function updates the graphic of the object specified by *ObjNo* and supports compressed data.

**Parameters:**

*ObjNo* Object Handle

*pSrc* Pointer to compressed source graphic

*CompressionType* Type of compression source graphic uses. This can be one of the `COMPRESSION_TYPE_*` predefined values.

**See also:**

[hel\\_ObjUpdateGfx](#), [hel\\_SwiUnComp](#)

**4.10.2.39 void ATTR\_DEPRECATED hel\_ObjUpdateInOAM (u8 *ObjNo*)**

Update an object in OAM immediately.

The [hel\\_ObjUpdateInOAM](#) function updates the object specified by `ObjNo` in OAM immediately.

**Parameters:**

*ObjNo* Object Handle

**Remarks:**

This function can be very handy though. Imagine you want, for example, change the visibility of an object immediately, instead of waiting until `ham_CopyObjToOAM` commits the new data to OAM, what usually happens when the VBL interrupt occurs (but this depends on your code). Then this function is what you want.

**Note:**

Check your local HAM documentation for more info about `ham_CopyObjToOAM`.

## 4.11 Object Rotation/Scaling Functions

### Functions

- THandle [hel\\_ObjRotScaleClone](#) (THandle Handle)  
*Clone a Rotation/Scaling Set.*
- THandle [hel\\_ObjRotScaleCreate](#) (u32 Angle, s32 ZoomX, s32 ZoomY)  
*Create a new Rotation/Scaling Set.*
- void [hel\\_ObjRotScaleDeInit](#) (void)  
*Deinitialize Object Rotation/Scaling SubSystem.*
- void [hel\\_ObjRotScaleDelete](#) (THandle Handle)  
*Delete a Rotation/Scaling Set.*
- u32 [hel\\_ObjRotScaleGetFreeSlots](#) (void)  
*Get amount of free Rotation/Scaling Sets.*
- void [hel\\_ObjRotScaleInit](#) (void)  
*Initialize Object Rotation/Scaling SubSystem.*
- void [hel\\_ObjRotScaleReset](#) (void)  
*Reset Object Rotation/Scaling SubSystem.*
- void [hel\\_ObjRotScaleTransmit](#) (void)  
*Transmit Rotation/Scaling Parameters to hardware.*
- void [hel\\_ObjRotScaleUpdate](#) (THandle Handle, u32 Angle, s32 ZoomX, s32 ZoomY)  
*Update Rotation/Scaling Parameters.*

### 4.11.1 Detailed Description

The Object Rotation/Scaling SubSystem can be used to create Rotation/Scaling Sets, which can be assigned to Objects, to perform rotation and scaling in hardware. The Rotation/Scaling Parameter-Attributes are shadowed internally and should be transmitted to hardware during VBL to avoid shearing.

Please see "demos/ObjRotation" for an example program.

### 4.11.2 Function Documentation

#### 4.11.2.1 THandle [hel\\_ObjRotScaleClone](#) (THandle *Handle*)

Clone a Rotation/Scaling Set.



The [hel\\_ObjRotScaleClone](#) function clones an existing Rotation/Scaling Set.

**Parameters:**

*Handle* Handle of Rotation/Scaling Set you want to clone

**Returns:**

Returns the Handle of the cloned Rotation/Scaling Set.

**4.11.2.2 THandle hel\_ObjRotScaleCreate (u32 Angle, s32 ZoomX, s32 ZoomY)**

Create a new Rotation/Scaling Set.

The [hel\\_ObjRotScaleCreate](#) creates a new Rotation/Scaling Set and returns a Handle to it.

**Parameters:**

*Angle* Rotation-angle in degree (0..359)

*ZoomX* Horizontal scaling

*ZoomY* Vertical scaling

**Returns:**

Returns a Handle to the created Rotation/Scaling Set.

ZoomX and ZoomY are in fixed point format, where 256 equals 1 and means no zoom at all. Bigger numbers enlarge, smaller reduce the size.

**See also:**

[hel\\_ObjRotScaleClone](#)

**4.11.2.3 void hel\_ObjRotScaleDeInit (void)**

Deinitialize Object Rotation/Scaling SubSystem.

Call [hel\\_ObjRotScaleInit](#) to deinitialize the Rotation/Scaling Parameter SubSystem.

**4.11.2.4 void hel\_ObjRotScaleDelete (THandle Handle)**

Delete a Rotation/Scaling Set.

The [hel\\_ObjRotScaleDelete](#) deletes the Rotation/Scaling Set specified by Handle when it's not referenced anymore.

**Parameters:**

*Handle* Handle of Rotation/Scaling Set you want to delete

In case the Rotation/Scaling Set has been cloned, it will not be delete until all clones have been deleted too.

#### 4.11.2.5 u32 [hel\\_ObjRotScaleGetFreeSlots](#) (void)

Get amount of free Rotation/Scaling Sets.

The [hel\\_ObjRotScaleGetFreeSlots](#) returns the amount of free/unused Rotation/Scaling Sets.

#### 4.11.2.6 void [hel\\_ObjRotScaleInit](#) (void)

Initialize Object Rotation/Scaling SubSystem.

Call [hel\\_ObjRotScaleInit](#) once to initialize the Rotation/Scaling Parameter SubSystem. This function must be called before using any other Object Rotation/Scaling function.

#### 4.11.2.7 void [hel\\_ObjRotScaleReset](#) (void)

Reset Object Rotation/Scaling SubSystem.

The [hel\\_ObjRotScaleReset](#) resets the Object Rotation/Scaling SubSystem.

#### 4.11.2.8 void [hel\\_ObjRotScaleTransmit](#) (void)

Transmit Rotation/Scaling Parameters to hardware.

The [hel\\_ObjRotScaleTransmit](#) forces the Rotation/Scaling SubSystem to copy its shadowed parameters to hardware (Vram). This function should be called during Vertical Blank.

#### 4.11.2.9 void [hel\\_ObjRotScaleUpdate](#) (THandle *Handle*, u32 *Angle*, s32 *ZoomX*, s32 *ZoomY*)

Update Rotation/Scaling Parameters.

[hel\\_ObjRotScaleUpdate](#) can be used to update the parameters of an existing Rotation/Scaling Set. This includes the rotation-angle and zoom.

##### Parameters:

*Handle* Handle of Rotation/Scaling Set

*Angle* Rotation-angle in degree

*ZoomX* Horizontal zoom

*ZoomY* Vertical zoom

##### See also:

[hel\\_ObjRotScaleCreate](#)

## 4.12 Pad Functions

The Pad-System is intended to indicate if a button is pressed or held. It supports two behaviours. The so called 'pressed state', which is either set when the button is pushed or released.

### Functions

- void [hel\\_PadInit](#) (void)  
*Initialize the pad system.*
- void [hel\\_PadReadState](#) (void)  
*Read pad state.*
- void [hel\\_PadReset](#) (void)  
*Reset the pad system.*
- void [hel\\_PadSetBehaviour](#) (u32 NewBehaviour)  
*Set Pad behaviour.*

### Variables

- TPad ATTR\_MEM\_IN\_EWRAM [hel\\_Pad](#)  
*Variable which holds the Pad states.*

#### 4.12.1 Detailed Description

The Pad-System is intended to indicate if a button is pressed or held. It supports two behaviours. The so called 'pressed state', which is either set when the button is pushed or released.

#### 4.12.2 Function Documentation

##### 4.12.2.1 void [hel\\_PadInit](#) (void)

Initialize the pad system.

Call this once at the beginning, to initialize the pad system. The pad behavior is set to PAD\_BEHAVIOUR\_PRESSED\_ON\_BUTTON\_UP.

See also:

[hel\\_PadReset](#), [hel\\_PadReadState](#)

#### 4.12.2.2 void `hel_PadReadState` (void)

Read pad state.

The `hel_PadReadState` function should be only called once per frame.

After calling the `hel_PadReadState` function, the global variable `hel_Pad` holds the current states.

**See also:**

[hel\\_PadInit](#), [hel\\_PadReset](#)

#### 4.12.2.3 void `hel_PadReset` (void)

Reset the pad system.

Call this to clear the pad states in the global variable `hel_Pad`. This also sets the pad behavior to `PAD_BEHAVIOUR_PRESSED_ON_BUTTON_UP`.

**See also:**

[hel\\_PadInit](#), [hel\\_PadReadState](#)

#### 4.12.2.4 void `hel_PadSetBehaviour` (u32 *NewBehaviour*)

Set Pad behaviour.

The Pad-Behaviour has only impact on the `Pressed` state. It can be specified if a pad button is pressed when it is pushed (button down) or released (button up).

**Parameters:**

*NewBehaviour* Specifies the new behaviour. This must be one of these defines:

- `PAD_BEHAVIOUR_PRESSED_ON_BUTTON_UP`
- `PAD_BEHAVIOUR_PRESSED_ON_BUTTON_DOWN`

**See also:**

[hel\\_PadReadState](#)

### 4.12.3 Variable Documentation

#### 4.12.3.1 TPad ATTR\_MEM\_IN\_EWRAM [hel\\_Pad](#)

Variable which holds the Pad states.

After [hel\\_PadReadState](#) has been called, the global variable `hel_Pad` holds the current pad states.

Here is an example program:

```
void main()
{
    // Initialize pad system
    hel_PadInit();

    // Set pad behavior
    hel_PadSetBehaviour(PAD_BEHAVIOUR_PRESSED_ON_BUTTON_DOWN);

    // Infinite loop
    while(TRUE)
    {
        // Make sure its a new frame
        if(IsNewFrame)
        {
            // Read pad states
            hel_PadReadState();

            // Access pad variable
            // Check if button down has been pressed
            if(hel_Pad.Pressed.Down)
            {
                Go_One_Menu_Item_Down();
            }
            else
            if(hel_Pad.Pressed.Up)
            {
                Go_One_Menu_Item_Up();
            }

            IsNewFrame = FALSE;
        }
    }

    // reset pad system
    hel_PadReset();
}
```

## 4.13 Palette Functions

### Functions

- void [hel\\_PalBgClear16](#) (u32 PaletteBank)  
*Set a background palette-bank to black.*
- void [hel\\_PalBgClear256](#) (void)  
*Set entire background palette to black.*
- void [hel\\_PalBgClearEx](#) (u32 StartIndex, u32 EndIndex, u32 R, u32 G, u32 B)  
*Set entire background palette to a specific color.*
- void [hel\\_PalBgInterpolate16](#) (void \*pPaletteA, void \*pPaletteB, u32 PaletteBank, u32 Step)  
*Interpolate between 16 color background palettes.*
- void [hel\\_PalBgInterpolate256](#) (void \*pPaletteA, void \*pPaletteB, u32 Step)  
*Interpolate between 256 color background palettes.*
- void [hel\\_PalBgInvert16](#) (u32 PaletteBank)  
*Invert colors of a 16 color background palette-bank.*
- void [hel\\_PalBgInvert256](#) (void)  
*Invert colors of entire background palette.*
- void [hel\\_PalBgInvertEx](#) (u32 StartIndex, u32 EndIndex)  
*Invert a part of background palette.*
- void [hel\\_PalBgLoad](#) (void \*SourceData, u32 NumColors)  
*Load a background palette.*
- void [hel\\_PalBgLoad16](#) (void \*SourceData, u32 PaletteBank)  
*Load a 16 color background palette.*
- void [hel\\_PalBgLoad16UnComp](#) (void \*SourceData, u32 PaletteBank, u32 CompressionType)  
*Load a 16 color compressed background palette.*
- void [hel\\_PalBgLoad256](#) (void \*SourceData)  
*Load a 256 color background palette.*
- void [hel\\_PalBgLoad256UnComp](#) (void \*SourceData, u32 CompressionType)  
*Load a 256 color compressed background palette.*
- void [hel\\_PalBgSave16](#) (void \*pDest, u32 PaletteBank)

*Save a 16 color background palette.*

- void [hel\\_PalBgSave256](#) (void \*pDest)

*Save a 256 color background palette.*

- void [hel\\_PalObjInterpolate16](#) (void \*pPaletteA, void \*pPaletteB, u32 PaletteBank, u32 Step)

*Interpolate between 16 color object palettes.*

- void [hel\\_PalObjInterpolate256](#) (void \*pPaletteA, void \*pPaletteB, u32 Step)

*Interpolate between 256 color object palettes.*

- void [hel\\_PalObjLoad](#) (void \*SourceData, u32 NumColors)

*Load an object palette.*

- void [hel\\_PalObjLoad16](#) (void \*SourceData, u32 PaletteBank)

*Load a 16 color object palette.*

- void [hel\\_PalObjLoad16UnComp](#) (void \*SourceData, u32 PaletteBank, u32 CompressionType)

*Load a 16 color compressed object palette.*

- void [hel\\_PalObjLoad256](#) (void \*SourceData)

*Load a 256 color object palette.*

- void [hel\\_PalObjLoad256UnComp](#) (void \*SourceData, u32 CompressionType)

*Load a 256 color compressed object palette.*

- void [hel\\_PalObjSave16](#) (void \*pDest, u32 PaletteBank)

*Save a 16 color object palette.*

- void [hel\\_PalObjSave256](#) (void \*pDest)

*Save a 256 color object palette.*

### 4.13.1 Detailed Description

Details...

The [Palette Functions](#) module provides functions to load Palettes into Vram, save from Vram, as well as functions to manipulate Palettes. Most functions use DMA Channel 3 in 16BIT mode to transfer the data either to or from Vram.

## 4.13.2 Function Documentation

### 4.13.2.1 void `hel_PalBgClear16` (u32 *PaletteBank*)

Set a background palette-bank to black.

The [hel\\_PalBgClear16](#) function sets the background palette-bank specified by *PaletteBank* to black.

**Parameters:**

*PaletteBank* The palette-bank (0..15)

**See also:**

[hel\\_PalBgClearEx](#), [hel\\_PalBgClear256](#)

### 4.13.2.2 void `hel_PalBgClear256` (void)

Set entire background palette to black.

The [hel\\_PalBgClear256](#) function sets the entire background palette to black.

**See also:**

[hel\\_PalBgClearEx](#), [hel\\_PalBgClear16](#)

### 4.13.2.3 void `hel_PalBgClearEx` (u32 *StartIndex*, u32 *EndIndex*, u32 *R*, u32 *G*, u32 *B*)

Set entire background palette to a specific color.

This function sets all palette entries between and including *StartIndex* to *EndIndex* to the color specified in *R*, *G* and *B*.

**Parameters:**

*StartIndex* Index to start at (0..254)

*EndIndex* Index to end at (*StartIndex*+1..n)

*R* Red amount (0..255)

*G* Green amount (0..255)

*B* Blue amount (0..255)

**See also:**

[hel\\_PalBgClear256](#), [hel\\_PalBgClear16](#)



#### 4.13.2.4 void `hel_PalBgInterpolate16` (void \* *pPaletteA*, void \* *pPaletteB*, u32 *PaletteBank*, u32 *Step*)

Interpolate between 16 color background palettes.

The [hel\\_PalBgInterpolate16](#) function interpolates between two 16 color background palettes and stores the interpolated RGB value into Vram.

**Parameters:**

*pPaletteA* Source colors

*pPaletteB* Target colors

*PaletteBank* Target palette bank of interpolated values. This can be between 0..15

*Step* Interpolation-Step. The *Step* can be between 0..31. 0 means colors are completely taken from *pPaletteA*, 31 means colors are completely taken from *pPaletteB*.

**Note:**

Interpolating between two palettes can be used to fake some kind of sunset for example. Use two palettes, one with colors how your level looks when it is day and one when it is night. Then simply interpolate between those palettes from level start to end. See sample project.

**See also:**

[hel\\_PalBgInterpolate256](#)

#### 4.13.2.5 void `hel_PalBgInterpolate256` (void \* *pPaletteA*, void \* *pPaletteB*, u32 *Step*)

Interpolate between 256 color background palettes.

The [hel\\_PalBgInterpolate256](#) function interpolates between two 256 color background palettes and stores the interpolated RGB value into Vram.

**Parameters:**

*pPaletteA* Source colors

*pPaletteB* Target colors

*Step* Interpolation-Step. The *Step* can be between 0..31. 0 means colors are completely taken from *pPaletteA*, 31 means colors are completely taken from *pPaletteB*.

**Note:**

Interpolating between two palettes can be used to fake some kind of sunset for example. Use two palettes, one with colors how your level looks when it is day and one when it is night. Then simply interpolate between those palettes from level start to end. See sample project. This function is rather slow, it requires about 20% execution time of one frame.

**See also:**

[hel\\_PalBgInterpolate16](#)

#### 4.13.2.6 void `hel_PalBgInvert16` (u32 *PaletteBank*)

Invert colors of a 16 color background palette-bank.

The [hel\\_PalBgInvert16](#) function inverts the colors of the background palette, specified by `PaletteBank`.

**Parameters:**

*PaletteBank* The palette-bank (0..15)

**See also:**

[hel\\_PalBgInvertEx](#), [hel\\_PalBgInvert256](#)

#### 4.13.2.7 void `hel_PalBgInvert256` (void)

Invert colors of entire background palette.

This function inverts the entire background palette entries (colors)

**See also:**

[hel\\_PalBgInvertEx](#), [hel\\_PalBgInvert16](#)

#### 4.13.2.8 void `hel_PalBgInvertEx` (u32 *StartIndex*, u32 *EndIndex*)

Invert a part of background palette.

This function inverts all background-palette colors between and including `StartIndex` and `EndIndex`.

**Parameters:**

*StartIndex* Index to start at (0..254)

*EndIndex* Index to end at (`StartIndex+1..n`)

**See also:**

[hel\\_PalBgClear256](#), [hel\\_PalBgClear16](#)

#### 4.13.2.9 void `hel_PalBgLoad` (void \* *SourceData*, u32 *NumColors*)

Load a background palette.

The [hel\\_PalBgLoad](#) function loads a background palette.

**Parameters:**

*SourceData* Pointer to palette-data

*NumColors* Number of colors to load. The destination starting index is 0.

**4.13.2.10 void hel\_PalBgLoad16 (void \* *SourceData*, u32 *PaletteBank*)**

Load a 16 color background palette.

The [hel\\_PalBgLoad16](#) function loads a 16 color background palette to the bank specified by `PaletteBank`

**Parameters:**

*SourceData* Pointer to palette-data

*PaletteBank* The palette-bank (0..15)

**Note:**

ham\_LoadBGPal16 does not work from time to time. I didn't find out the reason for that, so I just made [hel\\_PalBgLoad16](#) and use this one instead.

**4.13.2.11 void hel\_PalBgLoad16UnComp (void \* *SourceData*, u32 *PaletteBank*, u32 *CompressionType*)**

Load a 16 color compressed background palette.

The [hel\\_PalBgLoad256UnComp](#) function loads a compressed 16 color background palette

**Parameters:**

*SourceData* Pointer to palette-data

*PaletteBank* The palette-bank (0..15)

*CompressionType* Type of Compression. See `COMPRESSION_TYPE_*` predefined values.

**See also:**

[hel\\_PalBgLoad256UnComp](#), [hel\\_PalBgLoad16](#), [hel\\_SwiUnComp](#)

**4.13.2.12 void hel\_PalBgLoad256 (void \* *SourceData*)**

Load a 256 color background palette.

The [hel\\_PalBgLoad256](#) function loads a 256 color background palette

**Parameters:**

*SourceData* Pointer to palette-data

**4.13.2.13 void hel\_PalBgLoad256UnComp (void \* *SourceData*, u32 *CompressionType*)**

Load a 256 color compressed background palette.

The [hel\\_PalBgLoad256UnComp](#) function loads a compressed 256 color background palette

**Parameters:**

*SourceData* Pointer to palette-data

*CompressionType* Type of Compression. See `COMPRESSION_TYPE_*` predefined values.

**See also:**

[hel\\_PalBgLoad16UnComp](#), [hel\\_PalBgLoad256](#), [hel\\_SwiUnComp](#)

**4.13.2.14 void hel\_PalBgSave16 (void \*pDest, u32 PaletteBank)**

Save a 16 color background palette.

The [hel\\_PalObjSave16](#) function copies a 16 color background palette from VRAM to the memory location specified by `pDest`.

**Parameters:**

*pDest* Destination address, must point to an allocated buffer of at least 16 half-words (u16's).

*PaletteBank* Source palette-bank (0..15)

**4.13.2.15 void hel\_PalBgSave256 (void \*pDest)**

Save a 256 color background palette.

The [hel\\_PalBgSave256](#) function copies the 256 color background palette from VRAM to the memory location specified by `pDest`.

**Parameters:**

*pDest* Destination address, must point to an allocated buffer of at least 256 half-words (u16's).

**4.13.2.16 void hel\_PalObjInterpolate16 (void \*pPaletteA, void \*pPaletteB, u32 PaletteBank, u32 Step)**

Interpolate between 16 color object palettes.

The [hel\\_PalObjInterpolate16](#) function interpolates between two 16 color object palettes and stores the interpolated RGB value into Vram.

**Parameters:**

*pPaletteA* Source colors

*pPaletteB* Target colors

*PaletteBank* Target palette bank of interpolated values. This can be between 0..15

*Step* Interpolation-Step. The `Step` can be between 0..31. 0 means colors are completely taken from `pPaletteA`, 31 means colors are completely taken from `pPaletteB`.

See also:

[hel\\_PalObjInterpolate256](#)

#### 4.13.2.17 void [hel\\_PalObjInterpolate256](#) (void \* *pPaletteA*, void \* *pPaletteB*, u32 *Step*)

Interpolate between 256 color object palettes.

The [hel\\_PalObjInterpolate256](#) function interpolates between two 256 color object palettes and stores the interpolated RGB value into Vram.

**Parameters:**

*pPaletteA* Source colors

*pPaletteB* Target colors

*Step* Interpolation-Step. The *Step* can be between 0..31. 0 means colors are completely taken from *pPaletteA*, 31 means colors are completely taken from *pPaletteB*.

See also:

[hel\\_PalObjInterpolate16](#)

#### 4.13.2.18 void [hel\\_PalObjLoad](#) (void \* *SourceData*, u32 *NumColors*)

Load an object palette.

The [hel\\_PalObjLoad](#) function loads an object palette.

**Parameters:**

*SourceData* Pointer to palette-data

*NumColors* Number of colors to load. The destination starting index is 0.

#### 4.13.2.19 void [hel\\_PalObjLoad16](#) (void \* *SourceData*, u32 *PaletteBank*)

Load a 16 color object palette.

The [hel\\_PalObjLoad16](#) function loads a 16 color object palette to the bank specified by *PaletteBank*

**Parameters:**

*SourceData* Pointer to palette-data

*PaletteBank* The palette-bank (0..15)

**Note:**

`ham_LoadObjPal16` does not work from time to time. I didn't find out the reason for that, so I just made [hel\\_PalObjLoad16](#) and use this one instead.

#### 4.13.2.20 void `hel_PalObjLoad16UnComp` (void \* *SourceData*, u32 *PaletteBank*, u32 *CompressionType*)

Load a 16 color compressed object palette.

The [hel\\_PalObjLoad16UnComp](#) function loads a compressed 16 color object palette

##### Parameters:

*SourceData* Pointer to palette-data

*PaletteBank* The palette-bank (0..15)

*CompressionType* Type of Compression. See `COMPRESSION_TYPE_*` predefined values.

##### See also:

[hel\\_PalObjLoad256UnComp](#), [hel\\_PalObjLoad16](#), [hel\\_SwiUnComp](#)

#### 4.13.2.21 void `hel_PalObjLoad256` (void \* *SourceData*)

Load a 256 color object palette.

The [hel\\_PalObjLoad256](#) function loads a 256 color object palette

##### Parameters:

*SourceData* Pointer to palette-data

#### 4.13.2.22 void `hel_PalObjLoad256UnComp` (void \* *SourceData*, u32 *CompressionType*)

Load a 256 color compressed object palette.

The [hel\\_PalObjLoad256UnComp](#) function loads a compressed 256 color object palette

##### Parameters:

*SourceData* Pointer to palette-data

*CompressionType* Type of Compression. See `COMPRESSION_TYPE_*` predefined values.

##### See also:

[hel\\_PalObjLoad16UnComp](#), [hel\\_PalObjLoad256](#), [hel\\_SwiUnComp](#)

#### 4.13.2.23 void `hel_PalObjSave16` (void \* *pDest*, u32 *PaletteBank*)

Save a 16 color object palette.

The [hel\\_PalObjSave16](#) function copies a 16 color object palette from VRAM to the memory location specified by *pDest*.

**Parameters:**

*pDest* Destination address, must point to an allocated buffer of at least 16 half-words (u16's).

*PaletteBank* Source palette-bank (0..15)

**4.13.2.24 void hel\_PalObjSave256 (void \* pDest)**

Save a 256 color object palette.

The [hel\\_PalObjSave256](#) function copies the 256 color object palette from VRAM to the memory location specified by *pDest*.

**Parameters:**

*pDest* Destination address, must point to an allocated buffer of at least 256 half-words (u16's).

## 4.14 SRAM Functions

### Functions

- u8 ATTR\_DEPRECATED [hel\\_RAMEntryExists](#) (char \*pIdentifier)  
*Check if a SRAM entry exists.*

#### 4.14.1 Function Documentation

##### 4.14.1.1 u8 ATTR\_DEPRECATED [hel\\_RAMEntryExists](#) (char \*pIdentifier)

Check if a SRAM entry exists.

##### Parameters:

**pIdentifier** The identifier string by which the entry (data block) will be identified.

##### Returns:

Returns TRUE when the entry specified by pIdentifier exists in HAM's SRAM management, otherwise it returns FALSE.

This functions checks if the entry (data block) specified by pIdentifier exists in HAM's SRAM management.

##### See also:

Check ham\_InitRAM() in your local HAM documentation



## 4.15 Splash-Screen Functions

### Functions

- void [hel\\_Splash](#) (void \*pBitmap, void \*pPalette, u8 SplashTypeIn, u8 SplashTypeOut, u8 FadeInSpeed, u8 FadeOutSpeed, u8 WaitFrames, u8 AfterWaitFrames)

*Display a Splash-Screen.*

- void [hel\\_SplashUnComp](#) (void \*pBitmap, void \*pPalette, u8 SplashTypeIn, u8 SplashTypeOut, u8 FadeInSpeed, u8 FadeOutSpeed, u8 WaitFrames, u8 AfterWaitFrames, u32 CompressionType)

*Display a Splash-Screen with image decompression support.*

### 4.15.1 Function Documentation

#### 4.15.1.1 void [hel\\_Splash](#) (void \*pBitmap, void \*pPalette, u8 SplashTypeIn, u8 SplashTypeOut, u8 FadeInSpeed, u8 FadeOutSpeed, u8 WaitFrames, u8 AfterWaitFrames)

Display a Splash-Screen.

Please read the [hel\\_SplashUnComp](#) documentation.

See also:

[hel\\_SplashUnComp](#)

#### 4.15.1.2 void [hel\\_SplashUnComp](#) (void \*pBitmap, void \*pPalette, u8 SplashTypeIn, u8 SplashTypeOut, u8 FadeInSpeed, u8 FadeOutSpeed, u8 WaitFrames, u8 AfterWaitFrames, u32 CompressionType)

Display a Splash-Screen with image decompression support.

#### Parameters:

**pBitmap** Pointer to picture-data. The image must have one of the following sizes, depending on in what mode the GBA currently is:

- Mode 3 : 240\*160 pixels, 15bit BGR
- Mode 4 : 240\*160 pixels, 8 bit
- Mode 5 : 160\*120 pixels, 15bit BGR

This function also supports compressed image data. If the image data is compressed, the `CompressionType` parameter has to be set to the compression type it was compressed with.

**pPalette** Pointer to palette-data. In mode 3 and 5 set it to NULL

**SplashTypeIn** Type of splash screen in-fader. This can be one of these:

- `SPLASH_TYPE_BLACK`
- `SPLASH_TYPE_WHITE`

***SplashTypeOut*** Type of splash screen out-fader. This can be one of these:

- `SPLASH_TYPE_BLACK`
- `SPLASH_TYPE_WHITE`

***FadeInSpeed*** Speed of in-fading process. Setting this to 6 looks nice.

***FadeOutSpeed*** Speed of out-fading process. Setting this to 6 looks nice.

***WaitFrames*** How many frames the image should be displayed (without fade in/out)

***AfterWaitFrames*** How many frames the function should wait before leaving

***CompressionType*** The compression type of the data where `pBitmap` points to.  
This can be one of the `COMPRESSION_TYPE_*` predefined values.

**Note:**

The Splash functions are designed for bitmap modes (mode 3/4/5) only and uses the special effect register.

**See also:**

[hel\\_Splash](#), [hel\\_SwiUnComp](#)

## 4.16 BIOS Functions

### Functions

- void [hel\\_SwiLZ77UnCompVram](#) (void \*pDst, const void \*pSrc)  
*Expand LZ77-compressed data.*
- void [hel\\_SwiLZ77UnCompWram](#) (void \*pDst, const void \*pSrc)  
*Expand LZ77-compressed data.*
- void [hel\\_SwiRLUnCompVram](#) (void \*pDst, const void \*pSrc)  
*Expand Run-Length-Encoded data.*
- void [hel\\_SwiRLUnCompWram](#) (void \*pDst, const void \*pSrc)  
*Expand Run-Length-Encoded data.*
- void [hel\\_SwiUnComp](#) (void \*pDst, const void \*pSrc, u32 CompressionType)  
*Expand compressed data.*

### 4.16.1 Detailed Description

Description ...

The [BIOS Functions](#) Module can be used to execute functions from the GBA's BIOS. Several functions expect halfword or word-aligned memoryaddresses to work probably.

The debug library contains a memoryaddress-checking-mechanism which will display an error when a parameter contains an invalid memoryaddress. Invalid in sense of wrong alignment.

### 4.16.2 Function Documentation

#### 4.16.2.1 void [hel\\_SwiLZ77UnCompVram](#) (void \*pDst, const void \*pSrc)

Expand LZ77-compressed data.

##### Parameters:

**pDst** Destination address, must be halfword aligned!

**pSrc** Source address, must be word aligned and data size must be multiple 4.

#### 4.16.2.2 void [hel\\_SwiLZ77UnCompWram](#) (void \*pDst, const void \*pSrc)

Expand LZ77-compressed data.

**Parameters:***pDst* Destination address.*pSrc* Source address, must be word aligned and data size must be multiple 4.**4.16.2.3 void hel\_SwiRLUnCompVram (void \* *pDst*, const void \* *pSrc*)**

Expand Run-Length-Encoded data.

**Parameters:***pDst* Destination address, must be halfword aligned!*pSrc* Source address, must be word aligned and data size must be multiple 4.**4.16.2.4 void hel\_SwiRLUnCompWram (void \* *pDst*, const void \* *pSrc*)**

Expand Run-Length-Encoded data.

**Parameters:***pDst* Destination address.*pSrc* Source address, must be word aligned and data size must be multiple 4.**4.16.2.5 void hel\_SwiUnComp (void \* *pDst*, const void \* *pSrc*, u32 *CompressionType*)**

Expand compressed data.

**Parameters:***pDst* Destination address*pSrc* Source address*CompressionType* Compression-type of source data. This can be one of the following predefines values:

- COMPRESSION\_TYPE\_LZ77VRAM
- COMPRESSION\_TYPE\_LZ77WRAM
- COMPRESSION\_TYPE\_RLEVRAM
- COMPRESSION\_TYPE\_RLEWRAM

**Note:**

The destination and source addresses have to be aligned. Please see corresponding decompress function such as [hel\\_SwiLZ77UnCompVram](#), [hel\\_SwiLZ77UnCompWram](#), [hel\\_SwiRLUnCompVram](#), [hel\\_SwiRLUnCompWram](#)

## 4.17 System Control Functions

### Functions

- void [hel\\_SysSetPrefetch](#) (u32 Value)  
*Enable/Disable prefetch buffer.*

#### 4.17.1 Function Documentation

##### 4.17.1.1 void [hel\\_SysSetPrefetch](#) (u32 Value)

Enable/Disable prefetch buffer.

##### Parameters:

**Value** Set this either to TRUE or FALSE, whether you want to enable or disable the prefetch buffer.

##### Note:

For a detailed explanation what the prefetch buffer makes and for what it is good for, please read the documentation from. (links point to an external resource)

## 4.18 Tile Functions

Description.

### Functions

- void [hel\\_TileDeInit](#) (u32 BgNo)  
*Deinitialize a Tile-System.*
- void [hel\\_TileInit](#) (u32 BgNo, const u8 \*pTileData, u16 NumTilesRom, u16 \*pBufferA, u16 NumTilesRam, u16 \*pBufferB, u8 ColMode, u8 PalNo, u8 CbbOnlyMode)  
*Init a Tile-System.*
- u32 [hel\\_TileIsGraphicLoaded](#) (u32 BgNo, u32 RomTileNo)  
*Check if a graphic is loaded.*
- void [hel\\_TilePreloadGraphic](#) (u32 BgNo, u32 RomTileNo)  
*Preload a graphic.*
- void [hel\\_TileReleaseGraphic](#) (u32 BgNo, u32 RomTileNo)  
*Release a graphic.*
- void [hel\\_TileReloadGraphic](#) (u32 BgNo, u32 TargetRomTileNo, const u8 \*pSource)  
*Reload tile-graphic(s).*
- void [hel\\_TileReloadGraphic16](#) (u32 BgNo, u32 TargetRomTileNo, const u8 \*pSource)  
*Reload 4bit tile-graphic(s).*
- void [hel\\_TileReloadGraphic256](#) (u32 BgNo, u32 TargetRomTileNo, const u8 \*pSource)  
*Reload 8bit tile-graphic(s).*
- void [hel\\_TileShare](#) (u8 TargetBgNo, u8 SourceBgNo, u8 PalNo)  
*Share tilesetdata.*

### 4.18.1 Detailed Description

Description.

HEL's tile-system implementation dynamically reloads tile graphics and can therefore bypass the hardware limit of 1024 tiles which allows you to design more extended levels. The maximum amount of tiles what can be handled by HEL's tile-system is

32767 tiles. A few functions of HEL's tile-system are located in IWRAM, because they have to be fast. If you don't use dynamic tile reloading, these functions will not be linked into the ELF and therefore consume no extra memory.

HEL's tile-system does not dynamically allocate any memory from EWRAM. You have to pass allocated buffers instead. This option allows to use an own memory manager, in case it is necessary.

It can share tilesetdata between backgrounds. This is helpful if you have one (huge) tileset which is used by several backgrounds. When you share 16 color graphics, you can specify a different palettenumber for the shared data. This makes it possible to use the same tileset with different colors.

It comes with a function to check if specific tiles are loaded to videoram and reload their graphics to perform tileanimations.

16 and 256 color graphics are supported, all associated calculations are done for you automatically. You only have to pass the colormode when initializing the tile-system.

In debugmode it displays an errormessage when it tries to load a new tile into videoram but cannot find enough space.

**Note:**

Dynamic tile reloading with rotation maps is not supported yet.

## 4.18.2 Function Documentation

### 4.18.2.1 void hel\_TileDeInit (u32 BgNo)

Deinitialize a Tile-System.

The [hel\\_TileDeInit](#) function deinits the tilesystem for the background specified by **BgNo**.

**Parameters:**

**BgNo** Backgroundnumber to deininit the tilesystem from.

**See also:**

[hel\\_TileInit](#)

### 4.18.2.2 void hel\_TileInit (u32 BgNo, const u8 \* pTileData, u16 NumTilesRom, u16 \* pBufferA, u16 NumTilesRam, u16 \* pBufferB, u8 ColMode, u8 PalNo, u8 CbbOnlyMode)

Init a Tile-System.

**Parameters:**

**BgNo** Backgroundnumber to init the tilesystem for.

**pTileData** Pointer to source tilesetdata.

**NumTilesRom** Amount of tiles in the source tileset.

**pBufferA** A buffer of NumTilesRom allocated halfwords (u16's). pBufferA should be located in EWRAM.

**NumTilesRam** Amount of tiles you want to use in videoram. It's the maximum amount of tiles what can be displayed at the same time. For example the screen consists entirely of unique tiles, then NumTilesRam has to be set to 21\*31.

**pBufferB** A buffer of NumTilesRam\*3 allocated halfwords (u16's). pBufferB should be located in EWRAM.

**ColMode** Colormode of pTileData. Set it to 0 when using 4bit graphics, otherwise to 1.

**PalNo** When using 4bit graphics, set PalNo to palette number the graphic uses, otherwise set it to 0.

**CbbOnlyMode** Please consult your HAM documentation, see ham\_InitTileSet

```
#define RAM_SLOTS (224)

u16 BufferA[451] ATTR_MEM_IN_EWRAM;
u16 BufferB[RAM_SLOTS*3] ATTR_MEM_IN_EWRAM;

hel_TileInit
(
    0,                      // BgNo
    world_Tiles,            // Source GraphicData (Tileset)
    sizeof_16BIT(BufferA),  // Amount of tiles in GraphicData
    BufferA,                // BufferA
    RAM_SLOTS,              // Amount of tiles to use in videoram
    BufferB,                // BufferB
    1,                      // Colormode (1=256 Colors, 0=16 Colors)
    0,                      // Palettenumber (0..15)
    TRUE                    // CbbOnlyMode
);

// Create a Map here and enable dynamic tile reloading
// This can be done with hel_MapInit and hel_MapSetDynamicTileReloading
```

See also:

[hel\\_TileDeInit](#), [hel\\_TileShare](#), [hel\\_MapSetDynamicTileReloading](#)

#### 4.18.2.3 u32 hel\_TileIsGraphicLoaded (u32 BgNo, u32 RomTileNo)

Check if a graphic is loaded.

The [hel\\_TileIsGraphicLoaded](#) function can be used to check if a tile is currently loaded or not.

**Parameters:**

**BgNo** Backgroundnumber the tilesystem is assigned to

**RomTileNo** The tilenumber to check

**Returns:**

TRUE if loaded, otherwise FALSE



**4.18.2.4 void hel\_TilePreloadGraphic (u32 *BgNo*, u32 *RomTileNo*)**

Preload a graphic.

The [hel\\_TilePreloadGraphic](#) function loads the graphic for the tile referenced by `RomTileNo` to Vram.

**Parameters:**

***BgNo*** Backgroundnumber the tilesystem is assigned to

***RomTileNo*** The tilenumber in ROM to preload

**Note:**

Preloaded graphics must be manually released when they are no longer needed. Releasing the graphics must be done with [hel\\_TileReleaseGraphic](#).

**See also:**

[hel\\_TileReleaseGraphic](#)

**4.18.2.5 void hel\_TileReleaseGraphic (u32 *BgNo*, u32 *RomTileNo*)**

Release a graphic.

The [hel\\_TileReleaseGraphic](#) function releases the graphic for the tile referenced by `RomTileNo` in Vram.

**Parameters:**

***BgNo*** Backgroundnumber the tilesystem is assigned to

***RomTileNo*** The tilenumber in ROM to release

**Note:**

Only preloaded graphics must be manually released. If the graphic is used by another tile, it will not be released.

**See also:**

[hel\\_TilePreloadGraphic](#)

**4.18.2.6 void hel\_TileReloadGraphic (u32 *BgNo*, u32 *TargetRomTileNo*, const u8 \* *pSource*)**

Reload tile-graphic(s).

The [hel\\_TileReloadGraphic](#) can be used to reload the graphic for a given tile with either 4bit or 8bit colordepth.

**Parameters:**

***BgNo*** Backgroundnumber the tilesystem is assigned to

***TargetRomTileNo*** The tilenumber for what you want to reload the graphic.

*pSource* Pointer to source graphicdata.

**Note:**

Since [hel\\_TileReloadGraphic](#) must first check the colordepth to compute the correct datasize before loading the new tile to vram, this functions is slightly slower than the more specific reloading functions like [hel\\_TileReloadGraphic16](#) and [hel\\_TileReloadGraphic256](#). Before reloading a tile, first check if it is already loaded. In case it is not, you cannot reload it. To check if a tile is loaded, use [hel\\_TileIsGraphicLoaded](#).

**See also:**

[hel\\_TileReloadGraphic16](#), [hel\\_TileReloadGraphic256](#)

#### 4.18.2.7 void [hel\\_TileReloadGraphic16](#) (u32 *BgNo*, u32 *TargetRomTileNo*, const u8 \* *pSource*)

Reload 4bit tile-graphic(s).

The [hel\\_TileReloadGraphic16](#) can be used to reload the graphic for a given tile with 4bit colordepth (16 colors).

**Parameters:**

*BgNo* Backgroundnumber the tilesystem is assigned to

*TargetRomTileNo* The tilenumber for what you want to reload the graphic.

*pSource* Pointer to source graphicdata.

**Note:**

This function is faster than [hel\\_TileReloadGraphic](#). Before reloading a tile, first check if it is already loaded. In case it is not, you cannot reload it. To check if a tile is loaded, use [hel\\_TileIsGraphicLoaded](#).

**See also:**

[hel\\_TileReloadGraphic](#), [hel\\_TileReloadGraphic256](#)

#### 4.18.2.8 void [hel\\_TileReloadGraphic256](#) (u32 *BgNo*, u32 *TargetRomTileNo*, const u8 \* *pSource*)

Reload 8bit tile-graphic(s).

The [hel\\_TileReloadGraphic256](#) can be used to reload the graphic for a given tile with 8bit colordepth (16 colors).

**Parameters:**

*BgNo* Backgroundnumber the tilesystem is assigned to

*TargetRomTileNo* The tilenumber for what you want to reload the graphic.

*pSource* Pointer to source graphicdata.

**Note:**

This function is faster than [hel\\_TileReloadGraphic](#). Before reloading a tile, first check if it is already loaded. In case it is not, you cannot reload it. To check if a tile is loaded, use [hel\\_TileIsGraphicLoaded](#).

**See also:**

[hel\\_TileReloadGraphic](#), [hel\\_TileReloadGraphic16](#)

**4.18.2.9 void hel\_TileShare (u8 TargetBgNo, u8 SourceBgNo, u8 PalNo)**

Share tilesetdata.

The [hel\\_TileShare](#) function can be used to share tilesetdata. This is helpful if you have one tileset which is used by several backgrounds. Instead of initializing for each background an own TileSystem, you can (re)use the tilesetdata of an existing TileSystem. This has a big plus, because it requires less memory!

**Parameters:**

**TargetBgNo** Target, this is the backgroundnumber which receives the tilesetdata.

**SourceBgNo** Source, this is the backgroundnumber which provides the tilesetdata.

**PalNo** Palettenumber the target background should use. This makes it possible to use the same tilesetdata, but with different colors for each background. If you share 256 color graphics, set it to 0, otherwise to a palettenumber between 0 and 15.

When you no longer need the shared TileSystem, call [hel\\_TileDeInit](#).

**Note:**

If you share tilesetdata, please keep in mind it is used by at least two backgrounds. This means it will probably display more tiles from the same tileset and therefore the TileSystem needs more tileslots to work with. To increase the tileslots, use NumTilesRam and pBufferB parameter which you pass to [hel\\_TileInit](#), when initializing the source tileset.

**See also:**

[hel\\_TileInit](#), [hel\\_TileDeInit](#)

## 4.19 Timer Functions

### Functions

- u16 [hel\\_TimerGetValue](#) (u8 TimerNo)  
*Get Counter Value.*
- u8 [hel\\_TimerIsEnabled](#) (u8 TimerNo)  
*Check if timer is enabled.*
- void [hel\\_TimerResetValue](#) (u8 TimerNo)  
*Reset Counter Value.*
- void [hel\\_TimerSetEnable](#) (u8 TimerNo, u8 Value)  
*Start/Stop a timer.*
- void [hel\\_TimerSetFrequency](#) (u8 TimerNo, u8 Value)  
*Set frequency.*
- void [hel\\_TimerSetValue](#) (u8 TimerNo, u16 Value)  
*Set Counter Value.*
- void [hel\\_TimerStart](#) (u8 TimerNo)  
*Start a timer.*
- void [hel\\_TimerStop](#) (u8 TimerNo)  
*Stop a timer.*
- void [hel\\_TimerStopAll](#) (void)  
*Stop all Timers.*

### 4.19.1 Function Documentation

#### 4.19.1.1 u16 [hel\\_TimerGetValue](#) (u8 *TimerNo*)

Get Counter Value.

**Parameters:**

*TimerNo* Channel of the Timer (0..3)

**Returns:**

Returns the 16bit counter value of the timer specified by *TimerNo*

**See also:**

[hel\\_TimerSetValue](#)

#### 4.19.1.2 u8 `hel_TimerIsEnabled` (u8 *TimerNo*)

Check if timer is enabled.

Checks if the timer specified by `TimerNo` is enabled or not

**Parameters:**

*TimerNo* Channel of the Timer (0..3)

**Returns:**

Returns `TRUE` when the timer specified by `TimerNo` is enabled, otherwise `FALSE`.

**See also:**

[hel\\_TimerStart](#), [hel\\_TimerStop](#), [hel\\_TimerStopAll](#)

#### 4.19.1.3 void `hel_TimerResetValue` (u8 *TimerNo*)

Reset Counter Value.

The [hel\\_TimerResetValue](#) function sets the 16bit counter value of the timer specified by `TimerNo` to zero.

**Parameters:**

*TimerNo* Channel of the Timer (0..3)

**See also:**

[hel\\_TimerSetValue](#), [hel\\_TimerGetValue](#)

#### 4.19.1.4 void `hel_TimerSetEnable` (u8 *TimerNo*, u8 *Value*)

Start/Stop a timer.

Starts or stops the timer specified by `TimerNo`, depending on the value specified by `Value`

**Parameters:**

*TimerNo* Channel of the Timer (0..3)

*Value* Set `Value` to `TRUE`, to start the timer specified by `TimerNo`, `FALSE` to stop it.

**See also:**

[hel\\_TimerStart](#), [hel\\_TimerStop](#), [hel\\_TimerStopAll](#)

#### 4.19.1.5 void hel\_TimerSetFrequency (u8 *TimerNo*, u8 *Value*)

Set frequency.

Set the frequency at which the timer updates.

##### Parameters:

***TimerNo*** Channel of the Timer (0..3)

***Value*** The frequency value you want to set the timer to. This can be one of the following:

Value	Frequency
0	59,595 nanos (running 16,78MHz)
1	3,841 micrs (running clock/64)
2	15,256 micrs (running clock/256)
3	61,025 micrs (running clock/1024)

#### 4.19.1.6 void hel\_TimerSetValue (u8 *TimerNo*, u16 *Value*)

Set Counter Value.

The [hel\\_TimerSetValue](#) function sets the 16bit counter value of the timer specified by *TimerNo* to value specified by *Value*.

##### Parameters:

***TimerNo*** Channel of the Timer (0..3)

***Value*** Value to set the counter to

##### See also:

[hel\\_TimerGetValue](#)

#### 4.19.1.7 void hel\_TimerStart (u8 *TimerNo*)

Start a timer.

Starts the timer specified by *TimerNo*

##### Parameters:

***TimerNo*** Channel of the Timer (0..3)

##### See also:

[hel\\_TimerStop](#), [hel\\_TimerStopAll](#)

**4.19.1.8 void hel\_TimerStop (u8 *TimerNo*)**

Stop a timer.

Stops the timer specified by `TimerNo`

**Parameters:**

*TimerNo* Channel of the Timer (0..3)

**See also:**

[hel\\_TimerStart](#), [hel\\_TimerStopAll](#)

**4.19.1.9 void hel\_TimerStopAll (void)**

Stop all Timers.

The [hel\\_TimerStopAll](#) function stops all enabled timers.

**See also:**

[hel\\_TimerStop](#)

## 4.20 Window Functions

Description.

### Functions

- void [hel\\_WinDeInit](#) (u8 WinNo)  
*Set region of an existing window.*
- void [hel\\_WinHide](#) (u8 WinNo)  
*Hide an existing window.*
- void [hel\\_WinInit](#) (u8 WinNo, u8 Left, u8 Top, u8 Right, u8 Bottom, u8 Inside-Flags, u8 OutsideFlags)  
*Initializes a new Window.*
- void [hel\\_WinInitEx](#) (u8 WinNo, u8 Left, u8 Top, u8 Right, u8 Bottom, u8 Inside-Flags, u8 OutsideFlags, u8 SkipOutsideFlags)  
*Initializes a new Window.*
- void [hel\\_WinSetInsideFlags](#) (u8 WinNo, u8 InsideFlags)  
*Set inside flags of an existing window.*
- void [hel\\_WinSetOutsideFlags](#) (u8 WinNo, u8 OutsideFlags)  
*Set inside flags of an existing window.*
- void [hel\\_WinSetRegion](#) (u8 WinNo, u8 Left, u8 Top, u8 Right, u8 Bottom)  
*Set region of an existing window.*
- void [hel\\_WinSetVisible](#) (u8 WinNo, u8 Value)  
*Set visibility of an existing window.*
- void [hel\\_WinShow](#) (u8 WinNo)  
*Show an existing window.*
- void [hel\\_WinTransmit](#) (void)  
*Transmit Window Attributes to Vram.*

### 4.20.1 Detailed Description

Description.

The Window Feature can be used to split the screen into different regions. BG0-3, OBJ Layers and Special Effects can be separately enabled and/or disabled in each of these regions.



All functions, except for [hel\\_WinInitEx](#), get replaced with macros when you switch to release-mode. The Window Attributes are shadowed and need to be transmitted to hardware during Vertical Blank. This method avoids screen tearing!

## 4.20.2 Function Documentation

### 4.20.2.1 void [hel\\_WinDeInit](#) (u8 *WinNo*)

Set region of an existing window.

The [hel\\_WinSetRegion](#) function sets the region of an existing window.

**Parameters:**

*WinNo* Window-Number

**Example:**

```
// DeInit Window 0
hel_WinDeInit(0);
```

**See also:**

[hel\\_WinInit](#)

### 4.20.2.2 void [hel\\_WinHide](#) (u8 *WinNo*)

Hide an existing window.

The [hel\\_WinHide](#) function hides an existing window.

**Parameters:**

*WinNo* Window-Number

**Example:**

```
hel_WinHide
(
    0 // WinNo
);
```

**See also:**

[hel\\_WinInit](#), [hel\\_WinHide](#), [hel\\_WinSetVisible](#)

### 4.20.2.3 void [hel\\_WinInit](#) (u8 *WinNo*, u8 *Left*, u8 *Top*, u8 *Right*, u8 *Bottom*, u8 *InsideFlags*, u8 *OutsideFlags*)

Initializes a new Window.

The [hel\\_WinInit](#) function initializes a new window.

Please refer to [hel\\_WinInitEx](#) for details.

**Note:**

`hel_WinInit` just calls `hel_WinInitEx` with `SkipOutsideFlags` set to `FALSE`.

#### 4.20.2.4 `void hel_WinInitEx (u8 WinNo, u8 Left, u8 Top, u8 Right, u8 Bottom, u8 InsideFlags, u8 OutsideFlags, u8 SkipOutsideFlags)`

Initializes a new Window.

The `hel_WinInitEx` function initializes a new window. Windows can be used, for example, to split the screen into different regions.

**Parameters:**

**WinNo** Number of Window to initialize, this can be 0, 1 and 2. See notes!

**Left** Left screen-coordinate of window, specified in pixels.

**Top** Top screen-coordinate of window, specified in pixels.

**Right** Right screen-coordinate of window, specified in pixels.

**Bottom** Bottom screen-coordinate of window, specified in pixels.

**InsideFlags** The `InsideFlags` can be used to separately enable and/or disable BG0-3, OBJ layers as well as SpecialFX for the inner-region. The following values can be used and combined for the `InsideFlags` (See notes!):

- WIN\_BG0
- WIN\_BG1
- WIN\_BG2
- WIN\_BG3
- WIN\_OBJ
- WIN\_FX
- WIN\_ALL
- WIN\_NONE

**OutsideFlags** The `OutsideFlags` can be used to separately enable and/or disable BG0-3, OBJ layers as well as SpecialFX for the outer-region. Please note that window 0 and 1 share the `OutsideFlags`. You can only use the `OutsideFlags` for window 0 and 1 together. The following values can be used and combined for the `InsideFlags` (See notes!):

- WIN\_BG0
- WIN\_BG1
- WIN\_BG2
- WIN\_BG3
- WIN\_OBJ
- WIN\_FX
- WIN\_ALL
- WIN\_NONE

**SkipOutsideFlags** Since the `OutsideFlags` are shared for Window 0 and 1, you might don't want to override them when you init another window. When you set `SkipOutsideFlags` to `TRUE`, they will not be set! This parameter has no impact on Window 2.

**Note:**

Window 2, the Object Window, is specified by Objects (Sprites) which are having the Object-Mode set to OBJ\_MODE\_OBJWINDOW. Any pixels of these Objects with the palette index 0, are used as Object Window Region. The Object itself is not displayed! For more about Objects, please refer to the Object Documentation and read: [hel\\_ObjCreate](#)

OutsideFlags have to be specified always for Window 2 and SkipOutsideFlags have to be FALSE.

**Parameters:**

- WIN\_BG0 : Enable BG Layer 0
- WIN\_BG1 : Enable BG Layer 1
- WIN\_BG2 : Enable BG Layer 2
- WIN\_BG3 : Enable BG Layer 3
- WIN\_OBJ : Enable OBJ Layer
- WIN\_FX : Enable SpecialFX processing
- WIN\_ALL : Enable all BG Layers and OBJ Layer (cannot be combined)
- WIN\_NONE : Disable all BG Layers and OBJ Layer (cannot be combined)

**Example:**

```
hel_WinInitEx
(
    0,                // WinNo
    0,                // Left
    16,               // Top
    240,              // Right
    144,              // Bottom
    WIN_BG0 | WIN_BG 1 | WIN_FX, // InsideFlags
    WIN_NONE,         // OutsideFlags
    FALSE             // SkipOutsideFlags
);
```

**See also:**

[hel\\_WinSetRegion](#), [hel\\_WinDeInit](#)

**4.20.2.5 void hel\_WinSetInsideFlags (u8 WinNo, u8 InsideFlags)**

Set inside flags of an existing window.

The [hel\\_WinSetInsideFlags](#) function sets the InsideFlags of an existing window.

**Parameters:**

**WinNo** Window-Number

**InsideFlags** Please refer to [hel\\_WinInit](#)

**Example:**

```
hel_WinSetInsideFlags
(
    0,                // WinNo
    WIN_BG2 | WIN_FX  // InsideFlags
);
```

**See also:**

[hel\\_WinInit](#), [hel\\_WinSetOutsideFlags](#)

**4.20.2.6 void hel\_WinSetOutsideFlags (u8 WinNo, u8 OutsideFlags)**

Set inside flags of an existing window.

The [hel\\_WinSetInsideFlags](#) function sets the InsideFlags of an existing window.

**Parameters:**

**WinNo** Window-Number

**OutsideFlags** Please refer to [hel\\_WinInit](#)

**Example:**

```
hel_WinSetOutsideFlags
(
    0,                // WinNo
    WIN_ALL           // InsideFlags
);
```

**See also:**

[hel\\_WinInit](#), [hel\\_WinSetInsideFlags](#)

**4.20.2.7 void hel\_WinSetRegion (u8 WinNo, u8 Left, u8 Top, u8 Right, u8 Bottom)**

Set region of an existing window.

The [hel\\_WinSetRegion](#) function sets the region of an existing window.

**Parameters:**

**WinNo** Window-Number

**Left** Left screen-coordinate of window, specified in pixels.

**Top** Top screen-coordinate of window, specified in pixels.

**Right** Right screen-coordinate of window, specified in pixels.

**Bottom** Bottom screen-coordinate of window, specified in pixels.

**Example:**

```
hel_WinSetRegion
(
    0,      // WinNo
    0,      // Left
    16,     // Top
    240,    // Right
    144     // Bottom
);
```

See also:

[hel\\_WinInit](#), [hel\\_WinDeInit](#)

#### 4.20.2.8 void [hel\\_WinSetVisible](#) (u8 *WinNo*, u8 *Value*)

Set visibility of an existing window.

The [hel\\_WinSetVisible](#) function either shows or hides the Window specified by *WinNo*.

**Parameters:**

***WinNo*** Window-Number

***Value*** TRUE to show, FALSE to hide

**Example:**

```
hel_WinSetVisible
(
    0,      // WinNo
    FALSE   // Value
);
```

See also:

[hel\\_WinInit](#), [hel\\_WinShow](#), [hel\\_WinHide](#)

#### 4.20.2.9 void [hel\\_WinShow](#) (u8 *WinNo*)

Show an existing window.

The [hel\\_WinShow](#) function shows an existing window, which was made invisible from a previous call to [hel\\_WinHide](#).

**Parameters:**

***WinNo*** Window-Number

**Example:**

```
hel_WinShow
(
    0 // WinNo
);
```

See also:

[hel\\_WinInit](#), [hel\\_WinHide](#), [hel\\_WinSetVisible](#)

**4.20.2.10 void hel\_WinTransmit (void)**

Transmit Window Attributes to Vram.

The [hel\\_WinTransmit](#) function must be called during VBlank to transmit the shadowed window attributes to hardware.

## Chapter 5

# HEL Library Page Documentation

### 5.1 Improving compile time

You probably noticed the time to compile a rather large project is pretty time intensive. This is mostly because of the huge amount of graphic- and sound-data which get compiled everytime.

So the first thing you should do is to remove the `clean` call from the `all` target in your project `makefile`. This call is usually in each template and demo project which comes along with HAM. For small projects it does not make a big difference though.

The next evil thing I have seen so many times in the HAM forum and sourcecodes is to include graphics! Graphicdata is usually a lot and it does not make sense to include them. Now I hear you say 'why is it in most of the HAM demos then?'. Well, I tried to hold them as simple as possible, so you don't wonder where file `xy` comes from.

However, imagine you include 3MB data in `main.c`, now when you only change one letter in it, it includes all the graphics again and must compile them – that hurts!

It makes much more sense to compile them once and then only link them. This way they get only compiled when they have changed. To gain even for time, do not convert the graphics into a C array. GCC generates an assembly file from all C/C++ sources it compiles. So having it as a huge C array, it's first 'converted' into assembler code and then into an object file (`*.o`). The easiest way is to convert the graphics directly into assembler code. This works quite good and is a lot(!) faster than the C/C++ method.

Another option would be to `incbin` the RAW data. `incbin` is super fast! But this also has some drawbacks, because as far as I know, you must keep track of alignment and this could be a real pain. (if anyone knows how to auto-align them, without sending them through another toolchain to pad all RAW files, then please let me know, I'm very interested in it).

Hope it helps :)

## 5.2 Contact

Before submitting error-reports, please make sure it actually is an error in HEL. To check this, create a new project and test only the function in question. If the error still exists, send me the test-project with a detailed error-description.

Do not send the sourcecode of your entire demo or game. The test-project should be small and compile either on the HAM commandline or out of VisualHAM.

You can also post your question in the official [HAM forum](#), or if you want to make a bugreport or feature-request use the [Issuetracker](#).

However, if you wish to get in touch with me, find my email-address at [www.console-dev.de](http://www.console-dev.de)



## 5.3 Donate

If you wish to express your appreciation for the time I spend on developing HEL, writing example projects as well as this documentation, I accept and appreciate donations. Please use the PayPal button below to make your donation, thanks in advance!

## 5.4 FAQ - Frequently Asked Questions

**Q:** I made a demo/game using HEL, do you want it?

**A:** Of course!

**Q:** How you convert all the sample graphics?

**A:** The graphic conversions are all done with make. Open the makefile of a sample project and look for "gfx:" target. That's the graphic conversion command. If you want to execute it, open a HAM Shell ("External Tools | Command Prompt" in VisualHAM) and enter: `make gfx`

**Q:** In one of your sample projects I saw you use `GBACrusherCL.exe`, where can I get it?

**A:** The program is called GBA Crusher and it's written by John Sensebe. Use [google.com](http://google.com) or have a look at the [gbadev.org](http://gbadev.org) forum.

**Q:** Can I check how many memory my game uses?

**A:** You can use the following command to display the sizes of sections inside binary files (\*.elf;\*.a;\*.o;etc): `ham/gcc-arm/bin/arm-thumb-elf-size.exe -A game.elf`

**Q:** Are there any examples for HEL?

**A:** Yes, there are! The HEL zip archive you probably just downloaded contains them, just take a look at it. Hint: The folder is called `demos`

**Q:** My project needs so long to compile, what can I do?

**A:** See [Improving compile time](#)

**Q:** Why [Window Functions](#)? They are also covered by HAM!

**A:** [Window Functions](#) in HEL are implemented because:

- HAM's Window Functions do not support the SpecialFX attribute at this time (HAM 2.8)
- HAM's Window Functions do not work with Window 1 at this time (HAM 2.8)

**Q:** So I can use HEL Library for freeware/commercial/shareware projects?

**A:** Yes, you are free to use HEL Library as part of your program for any purpose including freeware, commercial and shareware programs, provided some credit is given to me. This can be in the credits-screen for example. Something like "HEL Library by Peter Schraut" is fine!

**Q:** Why do you have most of the Object functions implemented which are also available in HAM?

**A:** That's rather easy to explain. All object functions in HAM are 'real functions'. In HEL most of the functions get replaced with macros when you switch to `releasemode`. This is much much MUCH faster then! Make sure to take a look at the other FAQ entries related to debug/release things too.

**Q:** When I use the release library, the compiler complains it cannot find the necessary functions from the Object Module (`hel_Obj*`).

**A:** This is because I made a compiler directive switch to use macros instead of functions in releasemode. This is quite less overhead and has a huge performance improvement. So when you use the release library of HEL, you must open the file `ham/include/mygba.h` and comment out the line:

```
#define HAM_DEBUGGER
```

For more information on this please see [Debug and Release Library mystery](#)

**Q:** How can i speed-up my game?

**A:** In the debug library, HEL makes pretty much use ASSERT checking. This takes performance and space, -BUT- for the development process I do recommend to use it, because it helps a lot there, since it displays an errorscreen whenever you pass an invalid parameter to one of HEL's functions.

However, if you want to have full speed, use HEL's release library and go into `ham/include`, open `mygba.h` and comment out the `HAM_DEBUGGER` define. Then HEL uses for a lot of functions macros instead and that's really a speed boost ;) The registered HAM library is faster than the freeware version too, just as a side info. For more information on this please see [Debug and Release Library mystery](#)

**Q:** Debug and Release Library?

**A:** Please see [Debug and Release Library mystery](#)

**Q:** When I disable boundschecking using the `hel_MapSetBoundsCheck` function and scroll outside the map, screen corruption appears.

**A:** Well, that is why HEL has an option for it. When you disable BoundChecking, HEL does not check anymore if the current position of the map is still inside the (valid) mapdata. When it is not, then appears corruption on the screen. I recommend to leave it turned on.

**Q:** I want to use background-wrapping with my map, but it seems there is a bug.

**A:** This does not fit into HEL's concept and so it is not supported. For detailed information please follow [this](#) link and read my answer in the forum.

**Q:** NEW and FREE (malloc / free) are quite slow?

**A:** Oh well, dynamic memory allocation is slow. Try to avoid it! Use it only for initialization purposes, don't use it in your gameloop or similar things!

## 5.5 Installation

HEL Library comes in two forms. One for debug purposes, using [Assertion Checking](#) and one for the final release, which does not include `ASSERT's` and is faster and smaller.

Both libraries are called `libhel.a` and they are located in the `lib` directory from the HEL ZIP archive, which you probably just downloaded. You will see two subfolders in the `lib` directory, one is called `debug` and the other one is called `release`.

Copy `libhel.a` either from the `debug` or `release` directory into `ham/gcc-arm/lib` from your HAM installation.

Then you must also copy the HEL headerfile (`hel.h`) in order to get access to the functions. To do this, copy `hel.h` from HEL's `src` directory into the folder `ham/include` from your HAM installation.

Use one of the HEL demo projects, which come along with this archive, to test if HEL is working.

To link `libhel.a` to a project, you must add one line to the project makefile. Open the makefile from the project where you want to use HEL and add below `FILES` this line:

```
ADD_LIBS += $(GCCARM)/lib/libhel.a
```

Now it's ready to compile!

Please keep in mind, you must have HAM installed and `HAMlib` linked as well, otherwise HEL doesn't work right and spits out errors.

## 5.6 Introduction

**HEL library** is an additional function collection for the Gameboy Advance library called **HAMlib**.

HEL builds on the top of HAMlib. In order to use HEL, you have to link HAMlib. HEL introduces a nice set of functions what can make your coderlife a lot easier.

Most significant features of HEL include:

- Supports (in theory) unlimited large maps
- Supports parallax scrolling
- Supports datalayers (collision/event-maps)
- Supports dynamic tile reloading to bypass the hardware limit of only 1024 tiles
- Detailed API documentation and sample projects

HEL comes of course with a lot of more functions! Take a look at the function- reference. Most modules have a detailed description.

### Naming Convention:

HEL's naming convention makes it easy to find and remember functionnames.

Functions got their names using this system: [prefix] [module] [action]

```
[prefix] : The prefix is always hel_  
[module] : What Module/System from HEL you want to use. This can be Map/Obj/Dma and so forth.  
[action] : What the function is intended to do. Init for instance.
```

Using the above names, the complete functionname could be `hel_MapInit`.

This has a big advantage. When only writing `hel_Map` and then triggering your favorite IDE to popup the codecompletionlist, you have **all** map-functionnames from HEL available. This saves times when searching for functions!

### Sample Projects:

HEL comes with a few sample projects what demonstrate how to use the different modules of HEL. These can be found under 'hel/demos' and compile directly from the HAM shell or out of VisualHAM.

## 5.7 License

**HEL Library** (c) copyright 2003-2005 by Peter Schraut ([www.console-dev.de](http://www.console-dev.de))

HEL Library is distributed as **freeware**.

You are free to use HEL Library as part of your program for any purpose including freeware, commercial and shareware programs, provided some credit is given to the author.

The origin of this software must not be misrepresented; you must not claim your authorship. All redistributions must retain the original copyright notice and web site addresses.

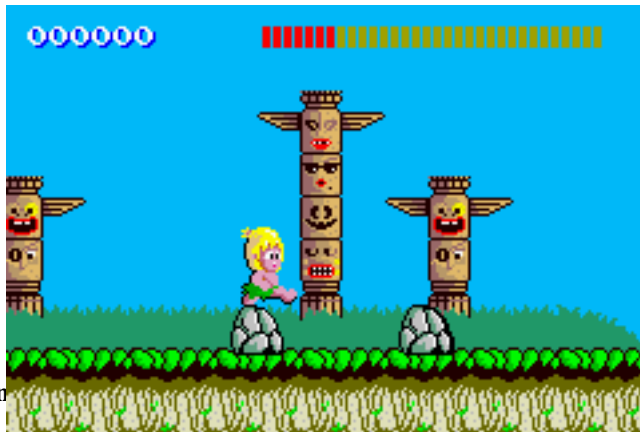
Commercial redistribution of the library is allowed only with an explicit written permission from the author.

If you finished a commercial game where HEL library is linked in, the author would like to be informed in what game it is used.

This software is provided 'as-is', without warranty of any kind, either expressed or implied. In no event shall the author be held liable for any damages arising from the use of this software.

## 5.8 Featured Projects

A selection of screenshots from real-world projects which are using HEL Library today.  
Please [Contact](#) me if you want to see your project here.



Won

<http://www.playeradvance.org/index.php?showtopic=13121>



Gho

<http://gbadev.org/index.php?ID=542>

Ago



<http://www.blindfold.z.pl>

Bro



<http://sevenravenstudio.altervista.org>

1942



<http://gbax.com/gbax2005/GBAX2005rr.html>





Fan

<http://fancyfair.free.fr>

Retu



<http://pokefight.free.fr/LboyPacoSofts/return.htm>



Mar

<http://www.playeradvance.org/index.php?showtopic=10480>



Flee

[href="http://gbax.com/gbax2005/GBAX2005rr.html"](http://gbax.com/gbax2005/GBAX2005rr.html)

## 5.9 Version History

- Version 1.7 (10th Oct 2005)
  - Fixed PDF documentation, images now included as well.
  - Fixed sample project "ObjWindow". Objects didn't move at all under GCC 4.0.1 when using the release library. Problem was `g_TotalFrames`, it was not declared as volatile.
  - Fixed sample project "Window". Window didn't move at all under GCC 4.0.1 when using the release library. Problem was `g_NewFrame`, it was not declared as volatile.
  - Added [On Assertion](#)
  - Added `hel_DebugSetOnAssert`. If you specify an "OnAssert" function, this one gets called before an assertion takes place.
  - Added "demos/GradientPalette", shows how to create a gradient background with only using one palette entry.
  - Added "tools/win32/paletti.exe", tool to create a palette from a bitmap file.
  - Fixed `HEL_ASSERT` ([Assertion Checking](#)), it didn't work with C++.
  - Fixed `HEL_ASSERT` ([Assertion Checking](#)) to work under latest DevKit-ARM (GCC 4.0.1)
  - Fixed `hel_MapSetParallaxRatio`. Changing the parallax ratio while scrolling caused it to not scroll at all.
  - Updated [Featured Projects](#)
  - Updated several sample projects to use `hel_IntrStopHandler`. Those are: Map, MapAnimatedTiles, MapDrawer, MapDynamicTileReloading, MapParallax, MapScrollTo, TextTyper, VirtualMap, Window
- Version 1.7-RC1 (4th Sep 2005)
  - Updated [Featured Projects](#)
  - Added `hel_BgSetPrio`
  - Fixed a bug in the map system which had impact on the background priority. The priority parameter of `ham_InitBg` had no impact after calling `hel_MapInit`. (thanks to RavenWorks for bugreport and test code)
  - Added [Object Rotation/Scaling Functions](#)
  - Added `hel_ObjRotScaleInit`
  - Added `hel_ObjRotScaleReset`
  - Added `hel_ObjRotScaleDeInit`
  - Added `hel_ObjRotScaleGetFreeSlots`
  - Added `hel_ObjRotScaleCreate`
  - Added `hel_ObjRotScaleClone`
  - Added `hel_ObjRotScaleDelete`
  - Added `hel_ObjRotScaleUpdate`

- Added [hel\\_ObjRotScaleTransmit](#)
  - Added [hel\\_PalObjInterpolate16](#)
  - Added [hel\\_PalObjInterpolate256](#)
  - Changed Window Feature System. The Window Attributes are shadowed now to avoid screen-tearing. Call [hel\\_WinTransmit](#) during Vertical Blank to transmit changes to hardware.
  - Added [hel\\_WinTransmit](#)
  - Added TWindowInfo structure
  - Added THandle type
  - Added HANDLE\_INVALID define (value = 0xff)
  - Added HANDLE\_ISVALID macro
- Version 1.6 (22nd Aug 2005)
    - Updated MapDynamicTileReloading samples project
    - Added memorymap see "lib/debug/memorymap.txt" and "lib/release/memorymap.txt"
    - Fixed [hel\\_WinSetVisible](#). Value parameter was missing (thanks to Paul C. for bugreport)
    - Marked a few functions as deprecated. These functions will probably get removed in the next major release (v2.0). If you use a deprecated function you will get the following warning: "warning: 'functionname' is deprecated"
    - Changed HEL\_ASSERT macro. It does not accept va\_args anymore and now all strings go into .rodata section. This makes a lot .text sections smaller. Only debug library affected.
    - Updated splash demos to use new graphic
    - Updated "demos/SharedMedia/gfx/splashhel.bmp" graphic
    - Included "katie", a tool to concatenate data. See "hel/tools/win32" directory.
    - Fixed lots of typos in documentation
    - Added [hel\\_TilePreloadGraphic](#)
    - Added [hel\\_TileReleaseGraphic](#)
    - Fixed a bug in [hel\\_ObjCountVisible](#)
    - Fixed a bug in [hel\\_ObjSendToBack](#). It had problems when passing the first object ([hel\\_ObjGetFirst](#)).
    - Moved 18 bytes from IWRAM into EWRAM
    - Enclosed couple of macros with curly braces to properly work with if expressions.
    - Added TPointSfp16 structure
    - Added TPointSfp32 structure
    - Added [hel\\_ObjGetShape](#)

- Added [hel\\_ObjGetSize](#)
- Added [hel\\_ObjGetSizeU16](#)
- Added [hel\\_ObjUpdateGfx](#)
- Added [hel\\_ObjUpdateGfxUnComp](#)
- Added ODD macro
- Added EVEN macro
- Added sample project InterpolatePalette
- Added [hel\\_PalBgInterpolate16](#)
- Added [hel\\_PalBgInterpolate256](#)
- Added RGB\_GET\_R macro
- Added RGB\_GET\_G macro
- Added RGB\_GET\_B macro
- Added sample project ExtractDigits. Shows how to use [hel\\_MathExtractDigits](#)
- Added [hel\\_MathExtractDigits](#)
- Added MEM\_OBJ\_SLOT define
- Added MEM\_OBJ\_SLOT\_PTR define
- Added OBJ\_SHAPE\_SQUARE define
- Added OBJ\_SHAPE\_HORIZONTAL define
- Added OBJ\_SHAPE\_VERTICAL define
- Added [hel\\_TileReloadGraphic16](#)
- Added [hel\\_TileReloadGraphic256](#)
- Updated [FAQ - Frequently Asked Questions](#)
- Fixed [hel\\_IntrStopHandler](#), DISPSTAT was not correctly updated (V-Blank, H-Blank, V-Counter). Errormessage was "you have the HBL-Interrupt turn on but not assigned to a function" (thanks to Francesco Marra for bugreport and testcode)
- Added [hel\\_SwiRLUnCompVram](#)
- Added [hel\\_SwiRLUnCompWram](#)
- Added [hel\\_SwiLZ77UnCompVram](#)
- Added [hel\\_SwiLZ77UnCompWram](#)
- Fixed Y character in assertion font
- Added [hel\\_PalObjLoad16UnComp](#)
- Added [hel\\_PalObjLoad256UnComp](#)
- Added ADDR\_WORDALIGNED macro
- Added ADDR\_HALFWORDALIGNED macro
- Added ADDR macros
- Added [hel\\_IntrStartHandler](#)
- Updated Interrupt sample project

- Added [hel\\_PalBgLoad256UnComp](#)
  - Added [hel\\_PalBgLoad16UnComp](#)
  - Added [hel\\_BmpLoad](#)
  - Added [hel\\_BmpLoadUnComp](#)
  - Added [hel\\_SplashUnComp](#)
  - Renamed sample project `SplashEx` to `Splash2`
  - Added new sample project `SplashLZ77UnComp`, shows how to display a splashscreen with LZ77 compressed imagedata.
  - Added new sample project `SplashRLUnComp`, shows how to display a splashscreen with RLE compressed imagedata.
  - Added [BIOS Functions](#)
  - Fixed [hel\\_SysSetPrefetch](#) undefined reference in release library
- Version 1.5 (8th Jul 2005)
    - Added [Featured Projects](#)
    - Moved 1024bytes from `.iwrwram` into `.rodata` section! This was only a problem in Debug library.
    - Added `MapDrawer` sample project
    - Updated/Fixed `hel_WinSetRegion` macro, to avoid warning in releasemode
    - Updated/Fixed `HEL_ASSERT` macro, to avoid warning when using Dev-KitARM
    - Updated [FAQ - Frequently Asked Questions](#)
    - Added returnvalue to [hel\\_MapBatchScrollBy](#) (thanks to Les Harris for sample code and function documentation)
    - Added `MapAnimatedTiles` demo project (thanks to Les Harris for helping me out here)
    - Added `MapDynamicTileReloading` demo project
    - Removed `ObjSetVisiableAll` demo, it's too obvious how to use this function.
    - Updated/Revised all demo projects
    - Graphics of demo projects now use shared resources. Shared graphics are located in "demos/SharedMedia".
    - Added [hel\\_MapSetDynamicTileReloading](#)
    - Added [hel\\_TileInit](#)
    - Added [hel\\_TileDeInit](#)
    - Added [hel\\_TileShare](#)
    - Added [hel\\_TileReloadGraphic](#)
    - Added [hel\\_TileIsGraphicLoaded](#)
    - Added `ATTR_DEPRECATED` macro
    - Added `ATTR_NOINLINE` macro



- Added `ATTR_FORCEINLINE` macro
- Added `ATTR_USED` macro
- Moved some variables from IWRAM to EWRAM
- Added [hel\\_MapJumpTo](#)
- Added `MAP_JUMPTO_LEFT`, `MAP_JUMPTO_RIGHT`, `MAP_JUMPTO_TOP` and `MAP_JUMPTO_BOTTOM` which can be used with [hel\\_MapJumpTo](#)
- Improved [hel\\_ObjClearOAM](#). Sets now also the background priority to 3 (behind all bg's) for every OAM entry.
- Updated [hel\\_ObjClearOAM](#) documentation
- Removed few assertion checks from [hel\\_MapGetTilePtrAt](#) to work with offscreen-coordinates too.
- Optimized [hel\\_CustomTextClear](#)
- Added [hel\\_PalBgLoad](#)
- Added [hel\\_PalObjLoad](#)
- Optimized [hel\\_MapGetPositionInPixelFrom](#)
- Added [hel\\_MapTransmitPosition](#)
- HEL now features an issuetracker: [HEL Issuetracker](#)
- Updated Text example. Located in folder: "demos/Text"
- Added new example project: "demos/TextTyper"
- Created lots of functions from the mapsystem as macros. This includes: [hel\\_MapInit](#), [hel\\_MapSetBoundsCheck](#), [hel\\_MapIsBoundsCheckEnabled](#), [hel\\_MapSetPosition](#), [hel\\_MapSetParallax](#), [hel\\_MapIsParallaxEnabled](#), [hel\\_MapSetCustomData](#), [hel\\_MapGetCustomData](#), [hel\\_MapSetScrollFlags](#), [hel\\_MapGetScrollFlags](#)
- Added [hel\\_MapSetScrollFlags](#)
- Added [hel\\_MapGetScrollFlags](#)
- Added `MAP_SCROLLFLAGS_LEFT`, `MAP_SCROLLFLAGS_RIGHT`, `MAP_SCROLLFLAGS_UP`, `MAP_SCROLLFLAGS_DOWN`, `MAP_SCROLLFLAGS_TRANSMITPOSITION`, `MAP_SCROLLFLAGS_DEFAULT`
- Optimized [hel\\_MapInit](#)
- Optimized [hel\\_MapInitEx](#)
- Updated documentation from a few map-functions
- Little performance increase in the mapsystem for rotationmaps
- Fixed a bug in the mapsystem when using rotationmaps
- Fixed a bug in the mapsystem when using maps which height is lesser than 31 tiles
- Added [hel\\_PalObjSave16](#)
- Added [hel\\_PalObjSave256](#)
- Added [hel\\_PalBgSave16](#)

- Added [hel\\_PalBgSave256](#)
  - Added [Debug Functions/Information](#) section
  - Added [Program execution breakpoints](#)
  - Updated [Introduction](#)
  - Fixed W character to ASSERT font
  - Added Questionmark character to ASSERT font
  - Optimized [hel\\_DmaCopy16](#)
  - Optimized [hel\\_DmaCopy32](#)
  - Optimized [hel\\_DmaSet16](#)
  - Optimized [hel\\_DmaSet32](#)
  - Optimized [hel\\_BmpClear](#)
  - Optimized [hel\\_Splash](#)
  - Added [hel\\_PalBgLoad16](#)
  - Added [hel\\_PalBgLoad256](#)
  - Added [hel\\_PalObjLoad256](#)
  - Optimized [hel\\_PalBgClear16](#)
  - Optimized [hel\\_PalBgClear256](#)
  - Optimized [hel\\_PalBgClearEx](#)
  - Added HEL\_DEBUG\_BRK, can be used so set debugging breakpoints for no\$gba!
- Version 1.4 (22nd May 2005)
    - Added WIN\_FX define
    - Added [hel\\_WinInit](#)
    - Added [hel\\_WinInitEx](#)
    - Added [hel\\_WinDeInit](#)
    - Added [hel\\_WinSetOutsideFlags](#)
    - Added [hel\\_WinSetInsideFlags](#)
    - Added [hel\\_WinSetRegion](#)
    - Added [hel\\_WinShow](#)
    - Added [hel\\_WinHide](#)
    - Added [hel\\_WinSetVisible](#)
    - Added example program ObjWindow, see folder: "Demos/ObjWindow"
    - Added example program Window, see folder: "Demos/Window"
    - undefined existing R\_WIN1Y define from HAM (mygba.h) and defined a new one which points to the correct memorylocation
    - Added [Debug Message Output](#)
    - Fixed [hel\\_SysSetPrefetch](#). Now it can also be turned off.
    - Added R\_BGXSCRLX define

- Added `R_BGXSCRLY` define
  - Added `PARALLAX_RATIO_QUARTER` define
  - Added `PARALLAX_RATIO_HALF` define
  - Added `PARALLAX_RATIO_THREEQUARTER` define
  - Fixed a bug in [hel\\_MapGetTileAt](#)
  - Updated the [FAQ - Frequently Asked Questions](#)
  - Improved [Assertion Checking](#) (Windows are now disabled while the ASSERT screen is displayed)
  - Added `HEL_LIBRARY_STRING` define
  - Added `HEL_DEBUG_PUT` define
  - Optimized [hel\\_FadePalette](#) and changed `AmountPerCall` handling. A value of 8 equals to a value of 1 before.
  - Updated Fade example program which comes with this archive, see folder: "Demos/Fade"
  - Recompiled all example programs with the new library version
  - Changed `WIN_SELECT`. Removed multiplications and using bit-shifts now
- Version 1.3 Final (28th Feb 2005)
  - Beautified documentation
  - Added [Assertion Checking](#) documentation
  - Updated the [FAQ - Frequently Asked Questions](#)
- Version 1.3 Beta 5 (17th Feb 2005)
  - Added [hel\\_PalObjLoad16](#)
  - Added a new feature to [hel\\_CustomTextPrint](#). You can now use `'\n'` to move over to a new line.
  - Added `FX_LAYER_BG0` define
  - Added `FX_LAYER_BG1` define
  - Added `FX_LAYER_BG2` define
  - Added `FX_LAYER_BG3` define
  - Added `FX_LAYER_OBJ` define
  - Added `FX_LAYER_BD` define
  - Added `FX_LAYER_ALL` define
  - Added `FX_LAYER_NONE` define
- Version 1.3 Beta 4 (13th Feb 2005)
  - Added [hel\\_ObjSetMode](#)
  - Added [hel\\_ObjGetPrio](#)

- Fixed a bug in [hel\\_MapRedraw](#). Thought i fixed it in beta 3 already, but there was still the same bug. Now everything should work fine ...
  - Further optimized the map-system. It's 11% faster for horizontal-scrolling and about 55% faster for vertical-scrolling. Both optimations were done for non rotation-maps only. HEL now uses DMA channel 3, in 16Bit mode, to update the map (when you scroll vertically). The improvement has impact on: [hel\\_MapScrollBy](#), [hel\\_MapBatchScrollBy](#), [hel\\_MapBatchScrollByEx](#), [hel\\_MapRedraw](#), [hel\\_MapScrollTo](#), [hel\\_MapSetPosition](#), [hel\\_MapSetPositionInPixel](#). The DMA updating was actually a bit trickier to implement than I thought, heh.
  - Fixed a bug in the map-system. This could be only noticed by analyzing the bg-maps with an emulator (bg viewer). Problem was it updated the invisible part of a map when the current map position + screenheight was greater than the mapsize, with values which didn't belong on the mapdata.
  - Added MID macro
  - Added LIMIT macro
  - Added ATTR\_MEM\_IN\_EWRAM, this makes the same as MEM\_IN\_EWRAM from "ham/include/mygba.h", but it does not produce an error when you use it with initialized data.
  - Added ATTR\_MEM\_IN\_IWRAM, this makes the same as MEM\_IN\_IWRAM from "ham/include/mygba.h", but it does not produce an error when you use it with initialized data.
  - Added ATTR\_FUNC\_IN\_IWRAM
  - Added null define, since until now there was only NULL
  - Added [hel\\_SysSetPrefetch](#)
  - Added [Improving compile time](#)
- Version 1.3 Beta 3 (17th Jan 2005)
    - Fixed a bug in [hel\\_MapRedraw](#). Since [hel\\_MapSetPosition](#) and [hel\\_MapSetPositionInPixel](#) internally use [hel\\_MapRedraw](#), these two functions were affected too. The resulting error message was: "Rowend must not be greater than mapheight in tiles"
    - Updated the [FAQ - Frequently Asked Questions](#)
  - Version 1.3 Beta 2 (?? ??? 2004)
    - Added [hel\\_ObjGetGfxSlot](#)
    - Added [hel\\_PadSetBehaviour](#)
    - Added PAD\_BEHAVIOUR\_PRESSED\_ON\_BUTTON\_UP define (Value: 0)
    - Added PAD\_BEHAVIOUR\_PRESSED\_ON\_BUTTON\_DOWN define (Value: 1)
    - Added [hel\\_ObjClearOAM](#)
    - Added [hel\\_ObjSendToBack](#)

- Added [hel\\_ObjSetPrio](#)
- Added SCREEN\_DISPLAY\_WIDTH define (Value: 240)
- Added SCREEN\_DISPLAY\_HEIGHT define (Value: 160)
- Added [hel\\_ObjSetXY](#)
- Added [hel\\_ObjSetY](#)
- Added [hel\\_ObjSetX](#)
- Added [hel\\_ObjSetVisible](#)
- Added [hel\\_BgSetMosaic](#)
- Added [hel\\_BgSetMosaicSize](#)
- Added [hel\\_ObjSetMosaicSize](#)
- Added an own ASSERT screen and macro called HEL\_ASSERT
- Improved the Interrupt Functions, read [hel\\_IntrStopHandler](#) documentation.
- Improved ASSERT message in [hel\\_IntrIsEnabled](#)
- Improved ASSERT message in [hel\\_IntrStopHandler](#)
- Added [hel\\_DmaSet8](#)
- Added [hel\\_DmaZeroMemory](#)
- Added more asserts in the DMA Module
- Updated [hel\\_DmaCopy16](#) documentation
- Updated [hel\\_DmaCopy32](#) documentation
- Updated [hel\\_DmaCopy16](#) documentation
- Updated [hel\\_DmaCopy32](#) documentation
- Added [hel\\_MapSetCustomData](#)
- Added [hel\\_MapGetCustomData](#)
- Added [hel\\_ObjSetVisibleAll](#)
- Added [hel\\_ObjHideAll](#)
- Added [hel\\_ObjShowAll](#)
- Added M\_BGXSCRLXY\_SET macro
- Fixed a tiny bug in the Map-System when using RotationMaps
- Added [hel\\_ObjCountVisible](#)
- Added [hel\\_ObjToggleVisible](#)
- Fixed assertion message in [hel\\_ObjBringToFront](#)
- Added more asserts in the Object Module, for more precise debugging
- Added F\_TIM0CNT\_IRQ\_ENABLED macro
- Added F\_TIM0CNT\_TIMER\_STARTEDmacro
- Added M\_TIM1CNT\_COUNTUP\_TIMING\_ENABLE macro
- Added M\_TIM1CNT\_COUNTUP\_TIMING\_DISABLE macro
- Added F\_TIM1CNT\_COUNTUP\_TIMING\_ENABLED macro

- Added F\_TIM1CNT\_IRQ\_ENABLED macro
  - Added F\_TIM1CNT\_TIMER\_STARTED macro
  - Added M\_TIM2CNT\_COUNTUP\_TIMING\_ENABLE macro
  - Added M\_TIM2CNT\_COUNTUP\_TIMING\_DISABLE macro
  - Added F\_TIM2CNT\_COUNTUP\_TIMING\_ENABLED macro
  - Added F\_TIM2CNT\_IRQ\_ENABLED macro
  - Added F\_TIM2CNT\_TIMER\_STARTED macro
  - Added M\_TIM3CNT\_COUNTUP\_TIMING\_ENABLE macro
  - Added M\_TIM3CNT\_COUNTUP\_TIMING\_DISABLE macro
  - Added F\_TIM3CNT\_COUNTUP\_TIMING\_ENABLED macro
  - Added F\_TIM3CNT\_IRQ\_ENABLED macro
  - Added F\_TIM3CNT\_TIMER\_STARTED macro
- Version 1.3 Beta 1 (16th Sep 2004)
  - Added [hel\\_MapGetPositionInPixelFrom](#)
  - Added [hel\\_MapBatchScrollByEx](#)
  - Added [hel\\_MapGetTilePtrAt](#)
  - Added [hel\\_MapGetTileAt](#)
  - Added [hel\\_MapInitVirtual](#)
  - Speed improvement in the map-system again
  - Added M\_BGXSCRLX\_SET macro
  - Added M\_BGXSCRLY\_SET macro
  - Added [FAQ - Frequently Asked Questions](#)
  - Fixed documentation. Links work again
- Version 1.2 (17th Jul 2004)
  - Added [hel\\_MapSetParallaxRatio](#)
  - Added [hel\\_MapSetParallax](#)
  - Added [hel\\_MapBatchScrollBy](#)
  - Added PARALLAX\_FLOAT\_TO\_RATIO macro
  - Added PARALLAX\_RATIO\_TO\_FLOAT macro
  - Added PARALLAX\_RATIO\_TO\_INT macro
  - Added PARALLAX\_RATIO\_ONE
  - Added new demo project "MapParallax"
  - Updated [Contact](#) text
  - Added [hel\\_IntrGetType](#)
- Version 1.1 (29th Jun 2004)

- Speedup in the map-system again! About 40% faster now!!! (for non rotation maps)
- Added `HEL_CURRENT_VERSION_STRING`
- Fixed `VBAOUT` macro
- Version 1.0 (29th Jun 2004)
  - Added [hel\\_ObjBringToFront](#)
  - Added [hel\\_ObjGetFirst](#)
  - Added [hel\\_BmpClear](#)
  - Added [hel\\_BmpGetBackBuffer](#)
- Version 0.9 (15th Feb 2004)
  - Performance BOOST in the map-system, it's around 50% faster now!!! (for non rotation maps)
  - Added a [Donate](#) Link
  - Added [hel\\_IntrStopHandler](#)
  - Added [hel\\_IntrStopAll](#)
  - Fixed [hel\\_CustomTextInit](#), the function-prototype has been changed, one parameter added for the *Palette Number*. You have to update your source-code in case you use [hel\\_CustomTextInit\(\)](#)!
  - Added Interrupt Demo
- Version 0.8 (26th Jan 2004)
  - Added [hel\\_FadeInit](#)
  - Added [hel\\_FadePalette](#)
  - Added [hel\\_FadeReset](#)
  - Added new example project, called "Fade"
  - Added [hel\\_RAMEntryExists](#)
  - Added [hel\\_IntrIsEnabled](#)
  - Added [hel\\_MapScrollTo](#), it's is like a camera. You can specify X/Y camera keypoints and the map scrolls automcatially to them.
  - Added new example project, called "MapScrollTo"
  - Added `VBAOUT` macro
  - Added `TPointU8`
  - Added `TPointU16`
  - Added `TPointU32`
  - Added `WIN_ALL` define
  - Added `WIN_NONE` define
  - Added `WIN_SELECT` define

- Version 0.7 (6th Dec 2003)
  - Updated [hel\\_MapSetBoundsCheck](#) documentation
  - Fixed NEW macro
  - Added [hel\\_Splash](#)
  - Added new example projects, Splash and SplashEx
  - Fixed a very strange bug in [hel\\_DmaSet16](#) and [hel\\_DmaSet32](#). The bug occurred when being in bitmap-mode, making a DmaSet and then switch to a tile-mode. The GBA just hang then :P
- Version 0.6 (26th Nov 2003)
  - Added NEW macro
  - Added FREE macro
  - Added [hel\\_ObjIsMosaic](#)
  - Added [hel\\_ObjIsRotScale](#)
  - Added EXTERN\_DATA8 macro
  - Added EXTERN\_DATA16 macro
  - Added EXTERN\_DATA32 macro
  - Changed [hel\\_MapScrollBy](#)'s return value. It returns now if it was able to scroll on x and/or y axis
  - Updated map example project
  - Updated [hel\\_MapSetCallbacks](#) documentation
  - Updated [hel\\_MapScrollBy](#) documentation
  - Added some version checking for better HAM 2.8 compatibility (some defines have same name)
- Version 0.5 (15th Nov 2003)
  - Fixed DMA\_NOCHANGE define
  - Added [hel\\_DmaSet16](#)
  - Added [hel\\_DmaSet32](#)
  - Forgot all the time to add the palette functions to hel.h, so there are a couple of new functions available now. Also changed "hel\_Palette..." into "hel\_Pal...".
  - Added [hel\\_PalBgInvert16](#)
  - Added [hel\\_PalBgInvert256](#)
  - Added [hel\\_PalBgInvertEx](#)
  - Added [hel\\_PalBgClear16](#)
  - Added [hel\\_PalBgClear256](#)
  - Added [hel\\_PalBgClearEx](#)
  - Added SET\_RECT macro



- Added [hel\\_ObjIsHFlip](#)
- Added [hel\\_ObjIsVFlip](#)
- Fixed [hel\\_ObjExists](#)
- Added [hel\\_ObjIsVisible](#)
- Version 0.4 (11th Nov 2003)
  - Fixed a bug in [hel\\_MapSetPositionInPixel](#) and [hel\\_MapSetPosition](#)
  - Fixed bug in [hel\\_MapScrollBy](#) when map is less than 32 tiles in height and boundcheck is enabled
  - [hel\\_MapScrollBy](#) returns now true/false, whether it was able to scroll or not
  - Removed parameter *UpdateVisibleOnly* from [hel\\_MapInitEx](#), because it wasn't used at all! (Added it for some testing and forgot to remove, sorry!)
  - Added [hel\\_ObjUpdateInOAM](#)
- Version 0.3 (27th Oct 2003)
  - Added [hel\\_MapInitEx](#)
  - Added DMA functions [hel\\_DmaCopy16](#), [hel\\_DmaCopy32](#)
  - Improved map system. It's now possible to specify different hardware-mapsizes instead of always using 32x32 tiles.
  - Fixed a bug in [hel\\_MapGetPositionInPixel](#) (Thanks to Christian Boutin for bug-report)
  - Fixed a bug in [hel\\_MapScrollBy](#) (Thanks to Christian Boutin for bug-report)
  - Added [hel\\_MapSetPositionInPixel](#)
  - Added some MAP\_\* defines
  - Fixed pad documentation
- Version 0.2 (6th Oct 2003)
  - Fixed rotation bug in mapfunctions
  - Fixed crash when no callback function is set in rotation-mode
  - Changed pMapData type in [hel\\_MapInit](#) from u16\* to void\*
  - Little speed improvement for map-functions
  - Added some compiler directives to check HAM version (CURRENT\_HAM\_VERSION\_\*)
  - Changed HEL\_VERSION\_MAJOR to HEL\_CURRENT\_VERSION\_MAJOR
  - Changed HEL\_VERSION\_MINOR to HEL\_CURRENT\_VERSION\_MINOR
  - Added some obj (wrapper)functions

- Version 0.1 (30th Sep 2003)
  - Initial release
  - Custom text system, to easily print text using own fonts
  - Pad control functions to check if a button is pressed or held
  - Map system with large map support, tested with a 512x512 tiles map

# Index

Background Functions, [14](#)

bg\_functions

hel\_BgSetMosaic, [14](#)

hel\_BgSetMosaicSize, [14](#)

hel\_BgSetPrio, [14](#)

BIOS Functions, [87](#)

Bitmap Mode Functions, [20](#)

bmp\_functions

hel\_BmpClear, [20](#)

hel\_BmpGetBackBuffer, [20](#)

hel\_BmpLoad, [20](#)

hel\_BmpLoadUnComp, [21](#)

Debug Functions/Information, [7](#)

DMA Functions, [22](#)

dma\_functions

hel\_DmaCopy16, [22](#)

hel\_DmaCopy32, [23](#)

hel\_DmaSet16, [23](#)

hel\_DmaSet32, [23](#)

hel\_DmaSet8, [24](#)

hel\_DmaZeroMemory, [24](#)

Fade Functions, [25](#)

fade\_functions

hel\_FadeInit, [25](#)

hel\_FadePalette, [25](#)

hel\_FadeReset, [26](#)

hel\_BgSetMosaic

bg\_functions, [14](#)

hel\_BgSetMosaicSize

bg\_functions, [14](#)

hel\_BgSetPrio

bg\_functions, [14](#)

hel\_BmpClear

bmp\_functions, [20](#)

hel\_BmpGetBackBuffer

bmp\_functions, [20](#)

hel\_BmpLoad

bmp\_functions, [20](#)

hel\_BmpLoadUnComp

bmp\_functions, [21](#)

hel\_CustomTextClear

text\_functions, [16](#)

hel\_CustomTextClearRow

text\_functions, [16](#)

hel\_CustomTextInit

text\_functions, [17](#)

hel\_CustomTextPrint

text\_functions, [17](#)

hel\_CustomTextPrintFmt

text\_functions, [18](#)

hel\_CustomTextType

text\_functions, [18](#)

hel\_DmaCopy16

dma\_functions, [22](#)

hel\_DmaCopy32

dma\_functions, [23](#)

hel\_DmaSet16

dma\_functions, [23](#)

hel\_DmaSet32

dma\_functions, [23](#)

hel\_DmaSet8

dma\_functions, [24](#)

hel\_DmaZeroMemory

dma\_functions, [24](#)

hel\_FadeInit

fade\_functions, [25](#)

hel\_FadePalette

fade\_functions, [25](#)

hel\_FadeReset

fade\_functions, [26](#)

hel\_IntrGetType

intr\_functions, [27](#)

hel\_IntrIsEnabled

intr\_functions, [28](#)

hel\_IntrStartHandler

intr\_functions, [28](#)

hel\_IntrStopAll

intr\_functions, [29](#)

hel\_IntrStopHandler

- intr\_functions, 29
- hel\_MapBatchScrollBy
  - map\_functions, 33
- hel\_MapBatchScrollByEx
  - map\_functions, 35
- hel\_MapDeInit
  - map\_functions, 35
- hel\_MapGetCustomData
  - map\_functions, 35
- hel\_MapGetPosition
  - map\_functions, 36
- hel\_MapGetPositionInPixel
  - map\_functions, 36
- hel\_MapGetPositionInPixelFrom
  - map\_functions, 37
- hel\_MapGetScrollFlags
  - map\_functions, 37
- hel\_MapGetTileAt
  - map\_functions, 37
- hel\_MapGetTilePtrAt
  - map\_functions, 38
- hel\_MapInit
  - map\_functions, 38
- hel\_MapInitEx
  - map\_functions, 39
- hel\_MapInitVirtual
  - map\_functions, 40
- hel\_MapIsBoundsCheckEnabled
  - map\_functions, 41
- hel\_MapIsParallaxEnabled
  - map\_functions, 41
- hel\_MapJumpTo
  - map\_functions, 41
- hel\_MapRedraw
  - map\_functions, 42
- hel\_MapScrollBy
  - map\_functions, 43
- hel\_MapScrollTo
  - map\_functions, 44
- hel\_MapSetBoundsCheck
  - map\_functions, 45
- hel\_MapSetCallbacks
  - map\_functions, 46
- hel\_MapSetCustomData
  - map\_functions, 46
- hel\_MapSetDynamicTileReloading
  - map\_functions, 47
- hel\_MapSetParallax
  - map\_functions, 47
- hel\_MapSetParallaxRatio
  - map\_functions, 47
- hel\_MapSetPosition
  - map\_functions, 48
- hel\_MapSetPositionInPixel
  - map\_functions, 48
- hel\_MapSetScrollFlags
  - map\_functions, 49
- hel\_MapTransmitPosition
  - map\_functions, 50
- hel\_MathExtractDigits
  - helMath, 51
- hel\_ObjBringToFront
  - obj\_functions, 55
- hel\_ObjClearOAM
  - obj\_functions, 55
- hel\_ObjCountVisible
  - obj\_functions, 55
- hel\_ObjCreate
  - obj\_functions, 55
- hel\_ObjCreate16
  - obj\_functions, 56
- hel\_ObjCreate256
  - obj\_functions, 56
- hel\_ObjExists
  - obj\_functions, 57
- hel\_ObjGetColorMode
  - obj\_functions, 57
- hel\_ObjGetFirst
  - obj\_functions, 57
- hel\_ObjGetGfxSlot
  - obj\_functions, 58
- hel\_ObjGetMode
  - obj\_functions, 58
- hel\_ObjGetPrio
  - obj\_functions, 58
- hel\_ObjGetShape
  - obj\_functions, 58
- hel\_ObjGetSize
  - obj\_functions, 59
- hel\_ObjGetSizeU16
  - obj\_functions, 59
- hel\_ObjHideAll
  - obj\_functions, 60
- hel\_ObjIsHFlip
  - obj\_functions, 60
- hel\_ObjIsMosaic
  - obj\_functions, 60
- hel\_ObjIsRotScale
  - obj\_functions, 61
- hel\_ObjIsVFlip

- obj\_functions, 61
- hel\_ObjIsVisible
  - obj\_functions, 61
- hel\_ObjRotScaleClone
  - helObjRotScale, 68
- hel\_ObjRotScaleCreate
  - helObjRotScale, 69
- hel\_ObjRotScaleDeInit
  - helObjRotScale, 69
- hel\_ObjRotScaleDelete
  - helObjRotScale, 69
- hel\_ObjRotScaleGetFreeSlots
  - helObjRotScale, 69
- hel\_ObjRotScaleInit
  - helObjRotScale, 70
- hel\_ObjRotScaleReset
  - helObjRotScale, 70
- hel\_ObjRotScaleTransmit
  - helObjRotScale, 70
- hel\_ObjRotScaleUpdate
  - helObjRotScale, 70
- hel\_ObjSelectRotScale
  - obj\_functions, 61
- hel\_ObjSendToBack
  - obj\_functions, 62
- hel\_ObjSetHFlip
  - obj\_functions, 62
- hel\_ObjSetMode
  - obj\_functions, 62
- hel\_ObjSetMosaicSize
  - obj\_functions, 63
- hel\_ObjSetPrio
  - obj\_functions, 63
- hel\_ObjSetRotScale
  - obj\_functions, 63
- hel\_ObjSetVFlip
  - obj\_functions, 64
- hel\_ObjSetVisible
  - obj\_functions, 64
- hel\_ObjSetVisibleAll
  - obj\_functions, 64
- hel\_ObjSetX
  - obj\_functions, 65
- hel\_ObjSetXY
  - obj\_functions, 65
- hel\_ObjSetY
  - obj\_functions, 65
- hel\_ObjShowAll
  - obj\_functions, 65
- hel\_ObjToggleVisible
  - obj\_functions, 66
- hel\_ObjUpdateGfx
  - obj\_functions, 66
- hel\_ObjUpdateGfxUnComp
  - obj\_functions, 66
- hel\_ObjUpdateInOAM
  - obj\_functions, 67
- hel\_Pad
  - pad\_functions, 72
- hel\_PadInit
  - pad\_functions, 71
- hel\_PadReadState
  - pad\_functions, 71
- hel\_PadReset
  - pad\_functions, 72
- hel\_PadSetBehaviour
  - pad\_functions, 72
- hel\_PalBgClear16
  - palette\_functions, 76
- hel\_PalBgClear256
  - palette\_functions, 76
- hel\_PalBgClearEx
  - palette\_functions, 76
- hel\_PalBgInterpolate16
  - palette\_functions, 76
- hel\_PalBgInterpolate256
  - palette\_functions, 77
- hel\_PalBgInvert16
  - palette\_functions, 77
- hel\_PalBgInvert256
  - palette\_functions, 78
- hel\_PalBgInvertEx
  - palette\_functions, 78
- hel\_PalBgLoad
  - palette\_functions, 78
- hel\_PalBgLoad16
  - palette\_functions, 78
- hel\_PalBgLoad16UnComp
  - palette\_functions, 79
- hel\_PalBgLoad256
  - palette\_functions, 79
- hel\_PalBgLoad256UnComp
  - palette\_functions, 79
- hel\_PalBgSave16
  - palette\_functions, 80
- hel\_PalBgSave256
  - palette\_functions, 80
- hel\_PalObjInterpolate16
  - palette\_functions, 80
- hel\_PalObjInterpolate256

- palette\_functions, 81
- hel\_PalObjLoad
  - palette\_functions, 81
- hel\_PalObjLoad16
  - palette\_functions, 81
- hel\_PalObjLoad16UnComp
  - palette\_functions, 81
- hel\_PalObjLoad256
  - palette\_functions, 82
- hel\_PalObjLoad256UnComp
  - palette\_functions, 82
- hel\_PalObjSave16
  - palette\_functions, 82
- hel\_PalObjSave256
  - palette\_functions, 83
- hel\_RAMEntryExists
  - sram\_functions, 84
- hel\_Splash
  - splashscreen\_functions, 85
- hel\_SplashUnComp
  - splashscreen\_functions, 85
- hel\_SwiLZ77UnCompVram
  - helSwi, 87
- hel\_SwiLZ77UnCompWram
  - helSwi, 87
- hel\_SwiRLUnCompVram
  - helSwi, 88
- hel\_SwiRLUnCompWram
  - helSwi, 88
- hel\_SwiUnComp
  - helSwi, 88
- hel\_SysSetPrefetch
  - sys\_functions, 89
- hel\_TileDeInit
  - helTile, 91
- hel\_TileInit
  - helTile, 91
- hel\_TileIsGraphicLoaded
  - helTile, 92
- hel\_TilePreloadGraphic
  - helTile, 92
- hel\_TileReleaseGraphic
  - helTile, 93
- hel\_TileReloadGraphic
  - helTile, 93
- hel\_TileReloadGraphic16
  - helTile, 94
- hel\_TileReloadGraphic256
  - helTile, 94
- hel\_TileShare
  - helTile, 95
- hel\_TimerGetValue
  - timer\_functions, 96
- hel\_TimerIsEnabled
  - timer\_functions, 96
- hel\_TimerResetValue
  - timer\_functions, 97
- hel\_TimerSetEnable
  - timer\_functions, 97
- hel\_TimerSetFrequency
  - timer\_functions, 97
- hel\_TimerSetValue
  - timer\_functions, 98
- hel\_TimerStart
  - timer\_functions, 98
- hel\_TimerStop
  - timer\_functions, 98
- hel\_TimerStopAll
  - timer\_functions, 99
- hel\_WinDeInit
  - win\_functions, 101
- hel\_WinHide
  - win\_functions, 101
- hel\_WinInit
  - win\_functions, 101
- hel\_WinInitEx
  - win\_functions, 102
- hel\_WinSetInsideFlags
  - win\_functions, 103
- hel\_WinSetOutsideFlags
  - win\_functions, 104
- hel\_WinSetRegion
  - win\_functions, 104
- hel\_WinSetVisible
  - win\_functions, 105
- hel\_WinShow
  - win\_functions, 105
- hel\_WinTransmit
  - win\_functions, 105
- helMath
  - hel\_MathExtractDigits, 51
- helObjRotScale
  - hel\_ObjRotScaleClone, 68
  - hel\_ObjRotScaleCreate, 69
  - hel\_ObjRotScaleDeInit, 69
  - hel\_ObjRotScaleDelete, 69
  - hel\_ObjRotScaleGetFreeSlots, 69
  - hel\_ObjRotScaleInit, 70
  - hel\_ObjRotScaleReset, 70
  - hel\_ObjRotScaleTransmit, 70

- hel\_ObjRotScaleUpdate, 70
- helSwi
  - hel\_SwiLZ77UnCompVram, 87
  - hel\_SwiLZ77UnCompWram, 87
  - hel\_SwiRLUnCompVram, 88
  - hel\_SwiRLUnCompWram, 88
  - hel\_SwiUnComp, 88
- helTile
  - hel\_TileDeInit, 91
  - hel\_TileInit, 91
  - hel\_TileIsGraphicLoaded, 92
  - hel\_TilePreloadGraphic, 92
  - hel\_TileReleaseGraphic, 93
  - hel\_TileReloadGraphic, 93
  - hel\_TileReloadGraphic16, 94
  - hel\_TileReloadGraphic256, 94
  - hel\_TileShare, 95
- Interrupt Functions, 27
- intr\_functions
  - hel\_IntrGetType, 27
  - hel\_IntrIsEnabled, 28
  - hel\_IntrStartHandler, 28
  - hel\_IntrStopAll, 29
  - hel\_IntrStopHandler, 29
- Map Functions, 31
- map\_functions
  - hel\_MapBatchScrollBy, 33
  - hel\_MapBatchScrollByEx, 35
  - hel\_MapDeInit, 35
  - hel\_MapGetCustomData, 35
  - hel\_MapGetPosition, 36
  - hel\_MapGetPositionInPixel, 36
  - hel\_MapGetPositionInPixelFrom, 37
  - hel\_MapGetScrollFlags, 37
  - hel\_MapGetTileAt, 37
  - hel\_MapGetTilePtrAt, 38
  - hel\_MapInit, 38
  - hel\_MapInitEx, 39
  - hel\_MapInitVirtual, 40
  - hel\_MapIsBoundsCheckEnabled, 41
  - hel\_MapIsParallaxEnabled, 41
  - hel\_MapJumpTo, 41
  - hel\_MapRedraw, 42
  - hel\_MapScrollBy, 43
  - hel\_MapScrollTo, 44
  - hel\_MapSetBoundsCheck, 45
  - hel\_MapSetCallbacks, 46
  - hel\_MapSetCustomData, 46
  - hel\_MapSetDynamicTileReloading, 47
  - hel\_MapSetParallax, 47
  - hel\_MapSetParallaxRatio, 47
  - hel\_MapSetPosition, 48
  - hel\_MapSetPositionInPixel, 48
  - hel\_MapSetScrollFlags, 49
  - hel\_MapTransmitPosition, 50
- Math Functions, 51
- obj\_functions
  - hel\_ObjBringToFront, 55
  - hel\_ObjClearOAM, 55
  - hel\_ObjCountVisible, 55
  - hel\_ObjCreate, 55
  - hel\_ObjCreate16, 56
  - hel\_ObjCreate256, 56
  - hel\_ObjExists, 57
  - hel\_ObjGetColorMode, 57
  - hel\_ObjGetFirst, 57
  - hel\_ObjGetGfxSlot, 58
  - hel\_ObjGetMode, 58
  - hel\_ObjGetPrio, 58
  - hel\_ObjGetShape, 58
  - hel\_ObjGetSize, 59
  - hel\_ObjGetSizeU16, 59
  - hel\_ObjHideAll, 60
  - hel\_ObjIsHFlip, 60
  - hel\_ObjIsMosaic, 60
  - hel\_ObjIsRotScale, 61
  - hel\_ObjIsVFlip, 61
  - hel\_ObjIsVisible, 61
  - hel\_ObjSelectRotScale, 61
  - hel\_ObjSendToBack, 62
  - hel\_ObjSetHFlip, 62
  - hel\_ObjSetMode, 62
  - hel\_ObjSetMosaicSize, 63
  - hel\_ObjSetPrio, 63
  - hel\_ObjSetRotScale, 63
  - hel\_ObjSetVFlip, 64
  - hel\_ObjSetVisible, 64
  - hel\_ObjSetVisibleAll, 64
  - hel\_ObjSetX, 65
  - hel\_ObjSetXY, 65
  - hel\_ObjSetY, 65
  - hel\_ObjShowAll, 65
  - hel\_ObjToggleVisible, 66
  - hel\_ObjUpdateGfx, 66
  - hel\_ObjUpdateGfxUnComp, 66

- hel\_ObjUpdateInOAM, 67
- Object Functions, 52
- Object Rotation/Scaling Functions, 68
- Pad Functions, 71
- pad\_functions
  - hel\_Pad, 72
  - hel\_PadInit, 71
  - hel\_PadReadState, 71
  - hel\_PadReset, 72
  - hel\_PadSetBehaviour, 72
- Palette Functions, 74
- palette\_functions
  - hel\_PalBgClear16, 76
  - hel\_PalBgClear256, 76
  - hel\_PalBgClearEx, 76
  - hel\_PalBgInterpolate16, 76
  - hel\_PalBgInterpolate256, 77
  - hel\_PalBgInvert16, 77
  - hel\_PalBgInvert256, 78
  - hel\_PalBgInvertEx, 78
  - hel\_PalBgLoad, 78
  - hel\_PalBgLoad16, 78
  - hel\_PalBgLoad16UnComp, 79
  - hel\_PalBgLoad256, 79
  - hel\_PalBgLoad256UnComp, 79
  - hel\_PalBgSave16, 80
  - hel\_PalBgSave256, 80
  - hel\_PalObjInterpolate16, 80
  - hel\_PalObjInterpolate256, 81
  - hel\_PalObjLoad, 81
  - hel\_PalObjLoad16, 81
  - hel\_PalObjLoad16UnComp, 81
  - hel\_PalObjLoad256, 82
  - hel\_PalObjLoad256UnComp, 82
  - hel\_PalObjSave16, 82
  - hel\_PalObjSave256, 83
- Splash-Screen Functions, 85
- splashscreen\_functions
  - hel\_Splash, 85
  - hel\_SplashUnComp, 85
- SRAM Functions, 84
- sram\_functions
  - hel\_RAMEntryExists, 84
- sys\_functions
  - hel\_SysSetPrefetch, 89
- System Control Functions, 89
- Text Functions, 16
- text\_functions
  - hel\_CustomTextClear, 16
  - hel\_CustomTextClearRow, 16
  - hel\_CustomTextInit, 17
  - hel\_CustomTextPrint, 17
  - hel\_CustomTextPrintFmt, 18
  - hel\_CustomTextType, 18
- Tile Functions, 90
- Timer Functions, 96
- timer\_functions
  - hel\_TimerGetValue, 96
  - hel\_TimerIsEnabled, 96
  - hel\_TimerResetValue, 97
  - hel\_TimerSetEnable, 97
  - hel\_TimerSetFrequency, 97
  - hel\_TimerSetValue, 98
  - hel\_TimerStart, 98
  - hel\_TimerStop, 98
  - hel\_TimerStopAll, 99
- win\_functions
  - hel\_WinDeInit, 101
  - hel\_WinHide, 101
  - hel\_WinInit, 101
  - hel\_WinInitEx, 102
  - hel\_WinSetInsideFlags, 103
  - hel\_WinSetOutsideFlags, 104
  - hel\_WinSetRegion, 104
  - hel\_WinSetVisible, 105
  - hel\_WinShow, 105
  - hel\_WinTransmit, 105
- Window Functions, 100