[1] [] 1 [Generated on Mon Aug 22 20:43:40 2005 for HEL Library by Doxygen] []Generated on Mon Aug 22 20:43:40 2005 for HEL Library by Doxygen

HEL Library Reference Manual

Generated by Doxygen 1.4.4

Mon Aug 22 20:43:40 2005

Contents

Chapter 1

HEL Library Documentation

Version:
1.6

Related Pages:

HEL function reference:

- Math Functions
- Object 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
  - 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:

Chapter 3

HEL Library Page Index

3.1 HEL Library Related Pages

Here is a list of all related documentation pages:

Chapter 4

HEL Library Module Documentation

4.1 Debug Functions/Information

Pages:

- Assertion Checking
- Debug Message Output
- Program execution breakpoints
- Debug and Release Library mystery

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.

4.2.1 Function Documentation

4.2.1.1 void hel_BgSetMosaic (u8 *BgNo*, u8 *Value*) [inline]

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*) [inline]

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

4.3 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 `pTextInfo` 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 *SizeU16*, 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

SizeU16 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 whereever 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 invisble 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 VirtualMaps!

```
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-curruption 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_
```

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_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_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 CompressionType)
  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 *DblSize*, 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
- `DblSize` = 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_SEMITRANSPARENT`
- `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 p 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: (ObjWidth∗ObjHeight)/2
- 4BIT: (ObjWidth∗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_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

4.10.2.23 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

4.10.2.24 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_SEMITRANSPARENT`

See also:

hel_ObjGetMode

4.10.2.25 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

4.10.2.26 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 between 0..3! It represents the background number where this object is placed before.

See also:

hel_ObjGetPrio

4.10.2.27 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

4.10.2.28 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

4.10.2.29 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

4.10.2.30 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

4.10.2.31 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

4.10.2.32 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

4.10.2.33 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

4.10.2.34 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

4.10.2.35 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

4.10.2.36 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

4.10.2.37 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 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.

4.11.1 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.

4.11.2 Function Documentation

4.11.2.1 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

4.11.2.2 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

4.11.2.3 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

4.11.2.4 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

4.11.3 Variable Documentation

4.11.3.1 TPad ATTR_MEM_IN_EWRAM hel_Pad

Variable which holds the Pad states.

After hel_PadReadState has been called, the golbal 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();
}
```

4.12 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_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.

4.12.1 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.

4.12.2 Function Documentation

4.12.2.1 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

4.12.2.2 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

4.12.2.3 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

4.12.2.4 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

4.12.2.5 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

4.12.2.6 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

4.12.2.7 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

4.12.2.8 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

4.12.2.9 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.

4.12.2.10 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.

4.12.2.11 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

4.12.2.12 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

4.12.2.13 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

4.12.2.14 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)

4.12.2.15 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).

4.12.2.16 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.

4.12.2.17 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.

4.12.2.18 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

4.12.2.19 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

4.12.2.20 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

4.12.2.21 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)

4.12.2.22 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).

4.13 SRAM Functions

Functions

- u8 ATTR_DEPRECATED hel_RAMEntryExists (char *pIdentifier)
  Check if a SRAM entry exists.

4.13.1 Function Documentation

4.13.1.1 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

4.14 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.

4.14.1 Function Documentation

4.14.1.1 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

4.14.1.2 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 on of these:

- SPLASH_TYPE_BLACK
- SPLASH_TYPE_WHITE

SplashTypeOut Type of splash screen out-fader. This can be on 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 display (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

4.15 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.

4.15.1 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.

4.15.2 Function Documentation

4.15.2.1 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.

4.15.2.2 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.

4.15.2.3 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.

4.15.2.4 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.

4.15.2.5 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

4.16 System Control Functions

Functions

- void hel_SysSetPrefetch (u32 Value)

  Enable/Disable prefetch buffer.

4.16.1 Function Documentation

4.16.1.1 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)

4.17 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.

4.17.1 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 rotatation maps is not supported yet.

4.17.2 Function Documentation

4.17.2.1 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 deinit the tilesystem from.

See also:
hel_TileInit

4.17.2.2 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

4.17.2.3 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

4.17.2.4 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

4.17.2.5 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

4.17.2.6 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

4.17.2.7 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

4.17.2.8 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

4.17.2.9 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

4.18 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.

4.18.1 Function Documentation

4.18.1.1 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

4.18.1.2 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

4.18.1.3 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

4.18.1.4 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

4.18.1.5 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)

4.18.1.6 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

4.18.1.7 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

4.18.1.8 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

4.18.1.9 void hel_TimerStopAll (void)

Stop all Timers.

The hel_TimerStopAll function stops all enabled timers.

See also:
hel_TimerStop

4.19 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 InsideFlags, u8 OutsideFlags)
  Initializes a new Window.
- void hel_WinInitEx (u8 WinNo, u8 Left, u8 Top, u8 Right, u8 Bottom, u8 InsideFlags, 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.

4.19.1 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. They are quite fast in this case, fast enough to be in a HBL interrupt for example.

4.19.2 Function Documentation

4.19.2.1 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

4.19.2.2 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

4.19.2.3 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`.

4.19.2.4 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 `SkipOutside-Flags` 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

4.19.2.5 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

4.19.2.6 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

4.19.2.7 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

4.19.2.8 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

4.19.2.9 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

Chapter 5

HEL Library Page Documentation

5.1 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

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;
}
```

5.2 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 interessted in it).

Hope it helps :)

5.3 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

5.4 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:

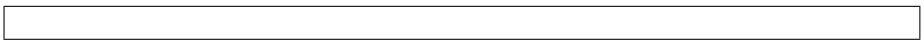
```
int main()
{
  ham_Init();

  // 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:



Note:

If you use HEL's breakpoint implementation with another emulator than NO$GBA, it will just leave the corresponsing opcode away, assuming you specified what emulator you use (see Debug Message Output) and therfore 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.

5.5 Debug and Release Library mystery

This document is intented 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 consumtion is archieved because it includes a few variables for reference-counting which are located in EWRAM ($<$ 500byte) and some self-modifing-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 errormessage 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

5.6 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 easiely 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_NOCHASH for NO$GBA support.

HEL_DEBUG_MSG_TYPE_NOCHASH 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 infront of you and scrolled a few lines down, you see:

```
//#define HEL_DEBUG_MSG_TYPE_VBA
#define HEL_DEBUG_MSG_TYPE_NOCHASH
```

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_NOCHASH
```

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

5.7 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!

5.8 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 or have a look at the 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 neccessary 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 infomation 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 if faster than the freeware version too, just as a side info. For more infomation 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, screencorruption 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!

5.9 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 `OFILES` 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 spites out errors.

5.10 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 functioname 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`.

5.11 License

5.12 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.

5.13 Version History

- Version 1.6
  - 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 proberly 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
  - Added Featured Projects
  - Moved 1024bytes from .iwram 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 DevKitARM
  - 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
  - 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
  - Beautified documentation
  - Added Assertion Checking documentation
  - Updated the FAQ - Frequently Asked Questions

- Version 1.3 Beta 5
  - 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
  - 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
  - Fixed a bug in hel_MapRedraw. Since hel_MapSetPosition and hel_-MapSetPositionInPixel internally use hel_MapRedraw, these two functions were affected too. The resulting errormessage was: "Rowend must not be greater than mapheight in tiles"
  - Updated the FAQ - Frequently Asked Questions

- Version 1.3 Beta 2
  - 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
  - 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
  - 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
  - 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
  - Added hel_ObjBringToFront
  - Added hel_ObjGetFirst

  - Added hel_BmpClear
  - Added hel_BmpGetBackBuffer

- Version 0.9
  - 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
  - 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 specifiy 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
  - Updated hel_MapSetBoundsCheck documention
  - 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 occured when being in bitmap-mode, making a DmaSet and then switch to a tile-mode. The GBA just hang then :P

- Version 0.6
  - 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
  - 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
  - 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
  - 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
  - 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
  - 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