New IMG includes  
Proposition of IMG include standard

Copyright © Imagination Technologies Ltd. All Rights Reserved.

‑‑This document is for internal use only. Neither the whole nor any part of the information contained in, nor the product described in, this document may be adapted or reproduced in any material form except with the written permission of Imagination Technologies Ltd. Distribution outside of Imagination Technologies Ltd is strictly forbidden.

Filename : New IMG includes.Proposition of IMG include standard.docx  
Version : 1.0.24 Not Issued - Live Document  
Issue Date : 07 Dec 2012  
Author : Imagination Technologies Ltd

Contents

Introduction 3

1.1. Goals 3

2. System Agnostic headers 3

2.1. Types (img\_types.h) 3

2.2. Definitions (img\_defs.h) 4

2.2.1. Different memory allocation 4

2.3. Error codes (img\_errors.h) 5

2.4. Pixel formats (img\_pixfmts.h) 5

2.5. Structures and enums (img\_structs.h) 5

2.5.1. Enumerations 6

2.6. General Include (img\_includes.h) 6

3. System specific headers 6

3.1. System types (img\_systypes.h) 6

3.2. System definitions (img\_sysdefs.h) 7

3.2.1. Asserting 8

3.2.2. Endianness 8

4. CMake behaviour 8

5. Tests 9

5.1. Compilation test 9

5.2. Unit tests 9

5.3. Running the tests 10

A. GNU/Linux kernel specific implementation 11

5.4. Memory management 11

5.4.1. Allocation priority 11

5.4.2. Different memory allocation 11

B. Porting code to the new IMG includes 12

List of Figures

**No table of figures entries found.**

IF{{Internal}}

Document History

| Issue | Date | Changes/Comments |
| --- | --- | --- |
|  | 19/09/12 | Creation |

END{{Internal}}

# Introduction

The legacy IMG include headers are getting out of hand. Some definitions are present for some platforms and not others, the types declarations are not 64 bits proof.

This document is a proposition on how the IMG includes should behave. It should be introduced with a working version of the new IMG includes that was applied to several branches of the code.

## Goals

The goal is to write some new IMG includes that are more organised and to remove all the elements that are not common from them. Doing so will make the set of headers non-compatible with existing code therefore this version should be used only for new projects. The interest is to have a defined behaviour for our macros (including the very-used IMG\_ASSERT) as well as good generic base for our different projects.

# System Agnostic headers

These headers are the one visible on from the user. They must be simple and CMake is used to determine the “system level” headers to use. They should not be affected by the particular system chosen.

All data types are upper case.

## Types (img\_types.h)

This header defines all the types that are based on integral types:

|  |  |
| --- | --- |
| Name | Description |
| IMG\_CHAR |  |
| IMG\_INT IMG\_UINT |  |
| IMG\_BOOL8 | This difference is needed for some projects (Boolean size is not defined) |
| IMG\_BOOL | This is a Boolean value (not bool in C++) of undefined size (using int) |
| IMG\_FLOAT IMG\_DOUBLE |  |
| IMG\_PINT8 IMG\_PUINT8  IMG\_PINT16  IMG\_PUINT16  IMG\_PINT32  IMG\_PUINT32  IMG\_INT64  IMG\_PUINT64  IMG\_PWCHAR | Pointer types |
| IMG\_HANDLE  IMG\_PVOID | Anonymous pointers (i.e. void\*). |
| IMG\_RESULT | Signed integer used for return code of functions. |

But it should also define some values:

|  |  |
| --- | --- |
| Name | Description |
| IMG\_NULL | Define as *NULL*. In C++ *NULL* is defined as 0 by the standard library. This is done as a define because *NULL* cannot be used with *typedef* |
| IMG\_TRUE  IMG\_FALSE | That should respect the C standard (i.e. 0 is false and that true is not false). IMG\_TRUE is fixed to 1.  This is unchanged in C++ (because IMG\_BOOL8 can be used too). |
| IMG\_VOID | A *typedef* of void are synonymous with the void keyword in C, but not in C++. |

## Definitions (img\_defs.h)

This covers different options using system definitions.

|  |  |
| --- | --- |
| Name | Description |
| IMG\_MALLOC(size) | Allocate some memory. C89 says: “*space is unitialized*.” |
| IMG\_CALLOC(nelem, size) | Allocate some memory. C89 says: “*space is initialized to zero bytes”* |
| IMG\_FREE(ptr) | Free some allocated memory. The pointer is not set back to IMG\_NULL. |
| IMG\_REALLOC(ptr, size) | Change the size of some allocation. C89 says: “*if the size is larger the new space is uninitialized”* |
| IMG\_UINT64\_TO\_UINT32(ui64) | checks that an ui64 can be used as ui32 using IMG\_ASSERT |

### Different memory allocation

There are different options of the memory allocation macros:

* Default: is using the system allocation and free (no defines) (see 3.2)
* Check: counts the number of calls to IMG\_MALLOC, IMG\_CALLOC and IMG\_FREE (this is NOT thread safe!)
* Test: use a global variable to know if IMG\_MALLOC or IMG\_CALLOC will fail (this is NOT thread safe!)

You can define both check and test together.

#### Check allocation

This is activated when IMG\_MALLOC\_CHECK is defined. It needs external variables to be added to your project:

|  |  |
| --- | --- |
| Name | Description |
| IMG\_UINT32 gui32Alloc; | The number of calls to IMG\_MALLOC and IMG\_CALLOC. This number is only increased if the allocation is a success. |
| IMG\_UINT32 gui32Free; | The number of calls to IMG\_FREE |

Warning: access to gui32Alloc and gui32Free is NOT thread safe.

#### Test allocation

This version can be used in order to force the failure of a particular memory allocation when performing tests. You need to define IMG\_MALLOC\_TEST. This version needs some external variables to be defined:

|  |  |
| --- | --- |
| Name | Description |
| IMG\_UINT32 gui32AllocFails; | To know if memory allocation should fail. The value must be set before calling IMG\_MALLOC or IMG\_CALLOC. The allocation will behave according to the value:   * if the value is 0 allocation succeeds * if value is 1 allocations fails (and values becomes 0) * if value is > 0 value is decremented and allocation succeeds   Basically this value is used to make the nth allocation fail. |

Warning: access to gui32AllocFails is NOT thread safe.

## Error codes (img\_errors.h)

This header contains the different error codes that can be returned using IMG\_RESULT.

Currently they are defined using pre-processor macros so they potentially can have negative values (enums are not allowed to have negative values).

In the future they could be cleaned (they are some duplicates). The errors should be fairly generic and do not include specific errors (each function should document its return codes, i.e. explain what happen if a specific error was returned).

Here are some used definitions:

|  |  |
| --- | --- |
| Name | Description |
| IMG\_SUCCESS | Returned when successful – should be 0 |
| IMG\_ERROR\_MALLOC\_FAILED | Returned when a memory allocation failed |
| IMG\_ERROR\_FATAL | Unspecified fatal error |
| IMG\_ERROR\_INVALID\_PARAMETERS | The provided parameters were incorrect |
| IMG\_ERROR\_UNEXPECTED\_STATE | Unexpected or illegal state found |
| IMG\_ERROR\_COULD\_NOT\_OBTAIN\_RESOURCE | A required resource could not be created or locked |
| IMG\_ERROR\_NOT\_INITIALISED | An attempt to access a structure/resource was made before it was initialised |
| IMG\_ERROR\_ALREADY\_INITIALISED | An attempt to initialise a structure/resource was made when it has already been initialised |
| IMG\_ERROR\_NOT\_SUPPORTED | The requested feature or mode is not supported |
| IMG\_ERROR\_MEMORY\_IN\_USE | Memory cannot be freed as it is still been used. |
| IMG\_ERROR\_VALUE\_OUT\_OF\_RANGE | A provided value exceeded stated bounds |

## Pixel formats (img\_pixfmts.h)

These should be discussed in another topic.

This header file should contain only the enum that lists all the pixel formats (if such approach is still the one used).

|  |  |
| --- | --- |
| IMG\_ePixelFormat | |
| Name | Description |
| IMG\_PIXFMT\_UNDEFINED | This format is considered invalid - used for default values into structures. |

## Structures and enums (img\_structs.h)

If a lot more enums should be shared then they should be moved to img\_enums.h.  
This header contains all shared structures and enums that are not dependent on one library to exist.

### Enumerations

The picture type (field or frame coded)

|  |  |
| --- | --- |
| IMG\_ePictureType | |
| Name | Description |
| IMG\_PICTYPE\_FRAME |  |
| IMG\_PICTYPE\_FIELD\_TOP  IMG\_PICTYPE\_FIELD\_BOTTOM |  |
| IMG\_PICTYPE\_PAIR |  |

The picture rotation or orientation (clockwise)

|  |  |
| --- | --- |
| IMG\_eOrientation | |
| Name | Description |
| IMG\_ROTATE\_NEVER |  |
| IMG\_ROTATE\_0  IMG\_ROTATE\_90  IMG\_ROTATE\_180  IMG\_ROTATE\_270 |  |

## General Include (img\_includes.h)

This file can (but doesn’t have to) be used to include all the previously mentioned headers.

# System specific headers

These headers are defining the system specific types and macros. Currently there are 3 platforms:

* C standard 99 (c99) that is compliant to this standard (i.e. used for GNU/Linux user mode)
* Microsoft (ms) that is windows specific
* Linux kernel (linux-kernel) that is specific to the internal headers for the kernel

Each of the system defines all the following files.

## System types (img\_systypes.h)

This header defines the sized data-types using the standard one available from the system (they have different names – and are in different headers – for different platforms).

|  |  |
| --- | --- |
| Name | Description |
| IMG\_WCHAR | Wide character (16b) |
| IMG\_INT8  IMG\_UINT8 | Signed and unsigned integers of different sizes |
| IMG\_INT16  IMG\_UINT16 |
| IMG\_INT32  IMG\_UINT32 |
| IMG\_INT64  IMG\_UINT64 |
| IMG\_BYTE | Atom of memory (usually similar to IMG\_UINT8) |
| IMG\_SIZE | Unsigned integer returned by *sizeof* operator (i.e. big enough to hold any memory allocation) (C89 equivalent *size\_t*). |
| IMG\_UINTPTR | Integer vairable that can hold a pointer value (C99 equivalent is *uintptr\_t*) |
| IMG\_PTRDIFF | Large enought to hold the signed difference of 2 pointer values (C89 equivalent *ptrdiff\_t*) |

## System definitions (img\_sysdefs.h)

This header defines some macros that are specific to the platform.

|  |  |
| --- | --- |
| Name | Description |
| IMG\_CONST | So that the usage of *const* can be disabled for some compilers |
| IMG\_INLINE | Because *inline* is different in C/C++ on win32 platforms |
| IMG\_LITTLE\_ENDIAN  IMG\_BIG\_ENDIAN | Values used to know the endianness.  See 3.2.2 |
| IMG\_BYTE\_ORDER | Is either IMG\_LITTLE\_ENDIAN or IMG\_BIG\_ENDIAN. Wouldn’t it be better if found at compilation or run time?  See 3.2.2 |
| IMG\_INT64PRFX | The *printf* like prefix to use when printing 64 bit values |
| IMG\_FTELL64 or IMG\_NO\_FTELL64 | IMG\_FTELL64 is the function to use for 64b support. If none exists IMG\_NO\_FTELL64 is defined. Not both should be defined at the same time. |
| IMG\_FSEEK64 or IMG\_NO\_FSEEK64 | Similar to IMG\_FTELL64 for fseek |
| IMG\_ALIGN(bytes) | Align elements to *bytes* bytes in memory. If not available on a platform leave empty. |

And also some memory operations:

|  |  |
| --- | --- |
| Name | Description |
| IMG\_MEMCPY(dest,src,size) |  |
| IMG\_MEMSET(ptr,val,size) |  |
| IMG\_MEMCMP(A,B,size) |  |
| IMG\_MEMMOVE(dest,src,size) |  |
| IMG\_SYSMALLOC(size) | System version of the macro defined in 2.2. These macros are used by the one in img\_defs.h to manage memory allocation.  If you are using the GNU/Linux kernel have a look at the appendix A. |
| IMG\_SYSCALLOC(nelem, size) |
| IMG\_SYSREALLOC(ptr, size) |
| IMG\_SYSFREE(ptr) |
| IMG\_STRDUP(ptr) |  |
| IMG\_STRCMP(A,B) |  |
| IMG\_STRCPY(dest,src) |  |
| IMG\_STRLEN(str) |  |

### Asserting

The macro IMG\_ASSERT(expected) must be defined to behave differently if the following macros are defined:

* NDEBUG: C89 says that “*if [it] is defined at the time <assert.h> is included the assert macro is ignored”*
* EXIT\_ON\_ASSERT: to allow asserts to test some code but do not stop the execution if the test fails.

The behaviour is therefore as follow:

|  |  |  |  |
| --- | --- | --- | --- |
| NDEBUG | | | |
|  |  | 0 | 1 |
| **EXIT\_ON\_ASSERT** | 0 | test||print | void |
|  | 1 | assert | assert |

Note that the 1/1 leads to an assert that is ignored.

### Endianness

Our current resolution of the endianness is relatively useless. For windows and linux it is assumed to be little endian.

Solutions exist to determine endianness at run time, like the following (that hasn’t been tested so cannot work for sure):

#define is\_bigendian() ( ((char)1) == 0 )

#define IMG\_BYTE\_ORDER (is\_bigendian() ? IMG\_BIG\_ENDIAN : IMG\_SMALL\_ENDIAN)

Or on GNU/Linux system the endian.h could be used.

In any case the endianness should be configurable in CMake rather than using the header file.

# CMake behaviour

The package defines the following variables:

* IMGINCLUDES\_INCLUDE\_DIRS
* IMGINCLUDES\_DEFINITIONS
* IMGINCLUDES\_LIBRARIES (empty)

The new IMG includes are configurable using CMake options. All these options must be set before searching for the package.

|  |  |  |
| --- | --- | --- |
| Name | Description | Default |
| IMGINCLUDES\_FORCE\_WIN32 | To force the usage of ms system | FALSE |
| IMGINCLUDES\_FORCE\_C99 | To force the usage of c99 system | FALSE |
| IMGINCLUDES\_FORCE\_KERNEL | To use the Linux kernel system | FALSE |
| IMGINCLUDES\_EXIT\_ON\_ASSERT | To exit when assert() fails (see 3.2.1) | TRUE |
| IMGINCLUDES\_MALLOC\_TEST | Choose allocation system, see 2.2.1 | FALSE |
| IMGINCLUDES\_MALLOC\_CHECK | Choose allocation system, see 2.2.1 | FALSE |

If you are not using any of the FORCE variables the system is chosen using the CMake variables WIN32 and UNIX. The only way to have the kernel definitions in CMake is to use IMGINCLUDES\_FORCE\_KERNEL.

For old CMake compatibility (and to help people move to the new includes) if the IMGINCLUDES\_OLD\_VARS is set to TRUE then the old CMake variables are defined:

* IMGINC\_INCLUDE\_DIRS
* IMGINC\_DEFINITIONS

# Tests

The compilation test is there to verify that the main keywords are defined. Their behaviour is tested in the unit tests.

The assert test can be run to verify the behaviour of the IMG\_ASSERT macro.

## Compilation test

Currently the following elements are missing in the compilation test:

Are missing from 2.1 Types (img\_types.h):

* IMG\_PINT8 IMG\_PUINT8
* IMG\_PINT16 IMG\_PUINT16
* IMG\_PINT32 IMG\_PUINT32
* IMG\_PINT64 IMG\_PUINT64
* IMG\_PWCHAR
* IMG\_PVOID

Only the following are tested from 2.3 Error codes (img\_errors.h):

* IMG\_SUCCESS
* IMG\_ERROR\_FATAL

All definitions from 2.4 Pixel formats (img\_pixfmts.h) are missing.

All definitions from 2.5 Structures and enums (img\_structs.h) are missing.

The generic include from 2.6 General Include (img\_includes.h) is not tested.

All definitions from 3.1 System types (img\_systypes.h) are tested.

Are missing from 3.2 System definitions (img\_sysdefs.h):

* IMG\_MEMMOVE

And the following are indirectly tested:

* IMG\_SYSMALLOC
* IMG\_SYSCALLOC
* IMG\_SYSREALLOC
* IMG\_SYSFREE

## Unit tests

All definitions from 2.1 Types (img\_types.h) are missing.

All definitions from 2.2 Definitions (img\_defs.h) are tested.

All definitions from 2.3 Error codes (img\_errors.h) are missing.

All definitions from 2.4 Pixel formats (img\_pixfmts.h) are missing.

All definitions from 2.5 Structures and enums (img\_structs.h) are missing.

The generic include from 2.6 General Include (img\_includes.h) is not tested.

Only the following define is missing from 3.1 System types (img\_systypes.h):

* IMG\_WCHAR

Only the following defines are tested from 3.2 System definitions (img\_sysdefs.h):

* IMG\_MEMCPY
* IMG\_MEMSET
* IMG\_MEMCMP

## Running the tests

The tests should be updated and run for each modification of the headers. This could be achieved through a build-server like Jenkins. Some examples of that are available on <http://devabuild01/> “Components – new IMG includes” projects.

# GNU/Linux kernel specific implementation

Several functions are slightly different in the kernel space in GNU/Linux.

## Memory management

### Allocation priority

When using memory allocation related in the GNU/Linux kernel a flag is needed to specify where the memory is taken from. The two most common flags are:

* GFP\_KERNEL is a normal priority allocation that can sleep, therefore the kernel can liberate memory if possible (unused pages etc).
* GFP\_ATOMIC: allocation has high priority and cannot sleep (use in interrupt handlers and bottom halves when using a spin lock). As the kernel cannot put the process to sleep it cannot swap pages etc to maybe make memory available. Therefore it has more chances to fail that a normal allocation. It should be used only in the contexts where memory allocation must be done very fast in a small amount.

The img\_sysdefs.h of the linux kernel defines IMG\_KMALLOC\_FLAG as GFP\_KERNEL. If your code needs high priority allocation in kernel mode you should consider a direct call to kmalloc() kcalloc() krealloc() or kstrdup().

As a result the memory allocation functions are slightly different than a simple macro. They check that the current context is not an interrupt context, if it is the case the allocation fails. For example the default IMG\_MALLOC used in linux kernel:

#define IMG\_SYSMALLOC(size) ({\

IMG\_VOID \*ptr = NULL; \

if (in\_interrupt()) { printk(KERN\_ERR "calling kmalloc in irq context is dangerous! %s %u", \_\_FILE\_\_, \_\_LINE\_\_); dump\_stack(); } \

else { \

ptr = kmalloc(size, IMG\_KMALLOC\_FLAG); \

} \

ptr; \

})

### Different memory allocation

As explained in 2.2.1 the IMG includes support different types of memory allocation. This should not be affected by the kernel memory management.

# Porting code to the new IMG includes

Is following a non-exhaustive list of changes that were done when porting the Felix drivers, TAL, MeOS and some IMGLIB to the new IMG includes:

The major modifications were:

* Changing CMake variables are IMG\_INCLUDES\_\* and not IMG\_INC\_\*
* changing lower case types to upper case (e.g. img\_uint8 to IMG\_UINT8)
* size\_t, ptrdiff\_t and uintptr\_t are defined as IMG\_SIZE, IMG\_PTRDIFF, IMG\_UINTPTR (no more IMG\_SIZE\_T that was IMG\_UINT32)
* headers are no longer including one and another if it is not needed (but a general header img\_include.h includes them all)  
  Sometime some #include were added to have the needed types/defs/errors
* some headers changed name or disappeared for consistency (e.g. img\_error.h is now img\_errors.h to be consistent with img\_types.h and img\_defs.h – img\_common.h disappeared as it was unclear what its content should be).
* IMG\_RESULT macros are uniform (all errors are IMG\_ERROR\_\*, eg IMG\_TIMEOUT becomes IMG\_ERROR\_TIMEOUT, similar for IMG\_FATAL\_ERROR, IMG\_MALLOC\_FAILED)
* IMG\_NULL is NULL in C and 0 in C++ - sometime IMG\_NULL was used as 0 or IMG\_FALSE
* No more pointer to functions in the IMG includes – they were moved to their respective libraries (e.g. IMG\_pfnEventCallback from img\_common.h becomes CBMAN\_pfnEventCallback in cbman.h)
* IMG\_BOOL are used as Boolean and do not have increment or decrement operations anymore.

Some projects need to define \_\_DO\_NOT\_BRIDGE\_\_ and \_\_PTR\_AND\_SIZE\_\_ in the headers that use those macro (as a rule of thumb all file defining rpc\_prefix should define them).

* + TAL (pdump\_cmds.h)
  + Kernel Component (xbuf\_comp.h, wrap\_utils.h, ibuf\_comp.h, page\_alloc.h, device\_io.h)
  + Portability framework (umisr\_api.h, dman\_api.h, ump\_api.h)