Contents

DDS

2/22/2020 • 2 minutes to read • Edit Online

The DirectDraw Surface file format (.dds) was introduced with DirectX 7 to store uncompressed and compressed (DXTn) textures. The file format supports mipmaps, cube maps, and volume maps. The DDS file format is supported by DirectXTex, DirectXTK, legacy D3DX, and other DirectX tools. Starting with Direct3D 10, DDS files support texture arrays.

> **NOTE**
>
> The D3DX (D3DX 9, D3DX 10, and D3DX 11) utility library is deprecated for Windows 8 and is not supported for Windows Store apps. We recommended that you make use of DirectXTex, DirectXTK, or both.

- Programming Guide for DDS
- Reference for DDS

Programming Guide for DDS

2/22/2020 • 5 minutes to read • Edit Online

Direct3D implements the DDS file format for storing uncompressed or compressed (DXTn) textures. The file format implements several slightly different types designed for storing different types of data, and supports single layer textures, textures with mipmaps, cube maps, volume maps and texture arrays (in Direct3D 10/11). This section describes the layout of a DDS file.

For help creating a texture in Direct3D 11, see How to: Create a Texture. For help in Direct3D 9, see Texture Support in D3DX (Direct3D 9).

- DDS File Layout
- DDS Variants
- Using Texture Arrays in Direct3D 10/11
- Common DDS File Resource Formats and Associated Header Content
- Related topics

DDS File Layout

A DDS file is a binary file that contains the following information:

- A DWORD (magic number) containing the four character code value 'DDS ' (0x20534444).

- A description of the data in the file.

  The data is described with a header description using **DDS_HEADER**; the pixel format is defined using **DDS_PIXELFORMAT**. Note that the **DDS_HEADER** and **DDS_PIXELFORMAT** structures replace the deprecated DDSURFACEDESC2, DDSCAPS2 and DDPIXELFORMAT DirectDraw 7 structures. **DDS_HEADER** is the binary equivalent of DDSURFACEDESC2 and DDSCAPS2. **DDS_PIXELFORMAT** is the binary equivalent of DDPIXELFORMAT.

  ```
  DWORD               dwMagic;
  DDS_HEADER          header;
  ```

  If the DDS_PIXELFORMAT dwFlags is set to DDPF_FOURCC and dwFourCC is set to "DX10" an additional **DDS_HEADER_DXT10** structure will be present to accommodate texture arrays or DXGI formats that cannot be expressed as an RGB pixel foramt such as floating point formats, sRGB formats etc. When the **DDS_HEADER_DXT10** structure is present the entire data description will looks like this.

  ```
  DWORD               dwMagic;
  DDS_HEADER          header;
  DDS_HEADER_DXT10    header10;
  ```

- A pointer to an array of bytes that contains the main surface data.

  ```
  BYTE bdata[]
  ```

- A pointer to an array of bytes that contains the remaining surfaces such as; mipmap levels, faces in a cube map, depths in a volume texture. Follow these links for more information about the DDS file layout for a:

texture, a cube map, or a volume texture.

```
BYTE bdata2[]
```

For broad hardware support, we recommend that you use the DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT_R8G8B8A8_UNORM_SRGB, DXGI_FORMAT_R8G8B8A8_SNORM, DXGI_FORMAT_B8G8R8A8_UNORM, DXGI_FORMAT_R16G16_SNORM, DXGI_FORMAT_R8G8_SNORM, DXGI_FORMAT_R8_UNORM, DXGI_FORMAT_BC1_UNORM, DXGI_FORMAT_BC1_UNORM_SRGB, DXGI_FORMAT_BC2_UNORM, DXGI_FORMAT_BC2_UNORM_SRGB, DXGI_FORMAT_BC3_UNORM, or DXGI_FORMAT_BC3_UNORM_SRGB format.

For more info about compressed texture formats, see Texture Block Compression in Direct3D 11 and Block Compression (Direct3D 10).

The D3DX library (for example, D3DX11.lib) and other similar libraries unreliably or inconsistently provide the pitch value in the **dwPitchOrLinearSize** member of the **DDS_HEADER** structure. Therefore, when you read and write to DDS files, we recommend that you compute the pitch in one of the following ways for the indicated formats:

- For block-compressed formats, compute the pitch as:

  max(1, ((width+3)/4)) * block-size

  The block-size is 8 bytes for DXT1, BC1, and BC4 formats, and 16 bytes for other block-compressed formats.

- For R8G8_B8G8, G8R8_G8B8, legacy UYVY-packed, and legacy YUY2-packed formats, compute the pitch as:

  ((width+1) >> 1) * 4

- For other formats, compute the pitch as:

  (width * bits-per-pixel + 7) / 8

  You divide by 8 for byte alignment.

> **NOTE**
>
> The pitch value that you calculate does not always equal the pitch that the runtime supplies, which is DWORD-aligned in some situations and byte-aligned in other situations. Therefore, we recommend that you copy a scan line at a time rather than try to copy the whole image in one copy.

DDS Variants

There are many tools that create and consume DDS files, but they can vary in the details of what they require in the header. Writers should populate the headers as fully as possible, and readers should check the minimal values for maximum compatibilty. To validate a DDS file, a reader should ensure the file is at least 128 bytes long to accommodate the magic value and basic header, the magic value is 0x20534444 ("DDS "), the DDS_HEADER size is 124, and the DDS_PIXELFORMAT in the header size is 32. If the DDS_PIXELFORMAT dwFlags is set to DDPF_FOURCC and a dwFourCC is set to "DX10", then the total file size needs to be at least 148 bytes.

There are some common variants in use where the pixel format is set to a DDPF_FOURCC code where dwFourCC is set to a D3DFORMAT or DXGI_FORMAT enumeration value. There is no way to tell if an enumeration value is a D3DFORMAT or a DXGI_FORMAT, so it is highly recommended that the "DX10" extension and DDS_HEADER_DXT10 header is used instead to store the dxgiFormat when the basic DDS_PIXELFORMAT cannot express the format.

The standard DDS_PIXELFORMAT should be preferred for maximum compatibilty to store RGB uncompressed data and DXT1-5 data as not all DDS tools support the DX10 extension.

Using Texture Arrays in Direct3D 10/11

The new DDS structures (DDS_HEADER and DDS_HEADER_DXT10) in Direct3D 10/11 extend the DDS file format to support an array of textures, which is a new resource type in Direct3D 10/11. Here is some sample code that shows how to access the different mipmap levels in an array of textures, using the new headers.

```
DWORD               dwMagic;
DDS_HEADER          header;
DDS_HEADER_DXT10    header10;

for (int iArrayElement = 0; iArrayElement < header10.arraySize; iArrayElement++)
{
   for (int iMipLevel = 0; iMipLevel < header.dwMipMapCount; iMipLevel++)
   {
     ...
   }
}
```

Common DDS File Resource Formats and Associated Header Content

RESOURCE FORMAT	DWFLAGS	DWRGBBITCOUNT	DWRBITMASK	DWGBITMASK	DWBBITMASK	DWABITMASK
DXGI_FORMAT_R8G8B8A8_UNORM D3DFMT_A8B8G8R8	DDS_RGBA	32	0xff	0xff00	0xff0000	0xff000000
DXGI_FORMAT_R16G16_UNORM D3DFMT_G16R16	DDS_RGBA	32	0xffff	0xffff0000		
** DXGI_FORMAT_R10G10B10A2_UNORM D3DFMT_A2B10G10R10	DDS_RGBA	32	0x3ff	0xffc00	0x3ff00000	
DXGI_FORMAT_R16G16_UNORM D3DFMT_G16R16	DDS_RGB	32	0xffff	0xffff0000		
DXGI_FORMAT_B5G5R5A1_UNORM D3DFMT_A1R5G5B5	DDS_RGBA	16	0x7c00	0x3e0	0x1f	0x8000

RESOURCE FORMAT	DWFLAGS	DWRGBBITCOUNT	DWRBITMASK	DWGBITMASK	DWBBITMASK	DWABITMASK
DXGI_FORMAT_B5G6R5_UNORM D3FMT_R5G6B5	DDS_RGB	16	0xf800	0x7e0	0x1f	
DXGI_A8_UNORM D3DFMT_A8	DDS_ALPHA	8				0xff
D3DFMT_A8R8G8B8	DDS_RGBA	32	0xff0000	0xff00	0xff	0xff000000
D3DFMT_X8R8G8B8	DDS_RGB	32	0xff0000	0xff00	0xff	
D3DFMT_X8B8G8R8	DDS_RGB	32	0xff	0xff00	0xff0000	
** D3DFMT_A2R10G10B10	DDS_RGBA	32	0x3ff00000	0xffc00	0x3ff	0xc0000000
D3DFMT_R8G8B8	DDS_RGB	24	0xff0000	0xff00	0xff	
D3DFMT_X1R5G5B5	DDS_RGB	16	0x7c00	0x3e0	0x1f	
D3DFMT_A4R4G4B4	DDS_RGBA	16	0xf00	0xf0	0xf	0xf000
D3DFMT_X4R4G4B4	DDS_RGB	16	0xf00	0xf0	0xf	
D3DFMT_A8R3G3B2	DDS_RGBA	16	0xe0	0x1c	0x3	0xff00
D3DFMT_A8L8	DDS_LUMINANCE	16	0xff			0xff00
D3DFMT_L16	DDS_LUMINANCE	16	0xffff			
D3DFMT_L8	DDS_LUMINANCE	8	0xff			
D3DFMT_A4L4	DDS_LUMINANCE	8	0xf			0xf0

RESOURCE FORMAT	DWFLAGS	DWFOURCC
DXGI_FORMAT_BC1_UNORM D3DFMT_DXT1	DDS_FOURCC	"DXT1"
DXGI_FORMAT_BC2_UNORM D3DFMT_DXT3	DDS_FOURCC	"DXT3"
DXGI_FORMAT_BC3_UNORM D3DFMT_DXT5	DDS_FOURCC	"DXT5"
* DXGI_FORMAT_BC4_UNORM	DDS_FOURCC	"BC4U"
* DXGI_FORMAT_BC4_SNORM	DDS_FOURCC	"BC4S"
* DXGI_FORMAT_BC5_UNORM	DDS_FOURCC	"ATI2"
* DXGI_FORMAT_BC5_SNORM	DDS_FOURCC	"BC5S"
DXGI_FORMAT_R8G8_B8G8_UNORM D3DFMT_R8G8_B8G8	DDS_FOURCC	"RGBG"
DXGI_FORMAT_G8R8_G8B8_UNORM D3DFMT_G8R8_G8B8	DDS_FOURCC	"GRGB"
* DXGI_FORMAT_R16G16B16A16_UNORM D3DFMT_A16B16G16R16	DDS_FOURCC	36
* DXGI_FORMAT_R16G16B16A16_SNORM D3DFMT_Q16W16V16U16	DDS_FOURCC	110
* DXGI_FORMAT_R16_FLOAT D3DFMT_R16F	DDS_FOURCC	111
* DXGI_FORMAT_R16G16_FLOAT D3DFMT_G16R16F	DDS_FOURCC	112
* DXGI_FORMAT_R16G16B16A16_FLOAT D3DFMT_A16B16G16R16F	DDS_FOURCC	113
* DXGI_FORMAT_R32_FLOAT D3DFMT_R32F	DDS_FOURCC	114

RESOURCE FORMAT	DWFLAGS	DWFOURCC
* DXGI_FORMAT_R32G32_FLOAT D3DFMT_G32R32F	DDS_FOURCC	115
* DXGI_FORMAT_R32G32B32A32_FLOAT D3DFMT_A32B32G32R32F	DDS_FOURCC	116
D3DFMT_DXT2	DDS_FOURCC	"DXT2"
D3DFMT_DXT4	DDS_FOURCC	"DXT4"
D3DFMT_UYVY	DDS_FOURCC	"UYVY"
D3DFMT_YUY2	DDS_FOURCC	"YUY2"
D3DFMT_CxV8U8	DDS_FOURCC	117
Any DXGI format	DDS_FOURCC	"DX10"

* = A robust DDS reader must be able to handle these legacy format codes. However, such a DDS reader should prefer to use the "DX10" header extension when it writes these format codes to avoid ambiguity.

** = Because of some long-standing issues in common implementations of DDS readers and writers, the most robust way to write out 10:10:10:2-type data is to use the "DX10" header extension with the DXGI_FORMAT code "24" (that is, the DXGI_FORMAT_R10G10B10A2_UNORM value). D3DFMT_A2R10G10B10 data should be converted to 10:10:10:2-type data before being written out as a DXGI_FORMAT_R10G10B10A2_UNORM format DDS file.

Related topics

D
D
S

DDS Texture Example

2/22/2020 • 2 minutes to read • Edit Online

For an uncompressed texture, use the DDSD_PITCH and DDPF_RGB flags; for a compressed texture, use the DDSD_LINEARSIZE and DDPF_FOURCC flags. For a mipmapped texture, use the DDSD_MIPMAPCOUNT, DDSCAPS_MIPMAP, and DDSCAPS_COMPLEX flags also as well as the mipmap count member. If mipmaps are generated, all levels down to 1-by-1 are usually written.

For a compressed texture, the size of each mipmap level image is typically one-fourth the size of the previous, with a minimum of 8 (DXT1) or 16 (DXT2-5) bytes (for square textures). Use the following formula to calculate the size of each level for a non-square texture:

```
max(1, ( (width + 3) / 4 ) ) x max(1, ( (height + 3) / 4 ) ) x 8(DXT1) or 16(DXT2-5)
```

This table lists the amount of space taken up by each layer for a 256-by-256 R8G8B8 texture, without using compression.

DDS COMPONENTS	# BYTES
header	128
256-by-256 main image	196608
128-by-128 mipmap image	49152
64-by-64 mipmap image	12288
32-by-32 mipmap image	3072
16-by-16 mipmap image	768
8-by-8 mipmap image	192
4-by-4 mipmap image	48
2-by-2 mipmap image	12
1-by-1 mipmap image	3

This table lists the amount of space taken up by each layer for the same texture using compression (DXT1).

DDS COMPONENTS	# BYTES
header	128
256-by-64 main image	8192

DDS COMPONENTS	# BYTES
128-by-32 mipmap image	2048
64-by-16 mipmap image	512
32-by-8 mipmap image	128
16-by-4 mipmap image	32
8-by-2 mipmap image	16
4-by-1 mipmap image	8
2-by-1 mipmap image	8
1-by-1 mipmap image	8

This table lists the amount of space taken up by each layer for the same texture using a DXGI compression format (in this case BC3_UNORM) that therefore requires the extended header:

DDS COMPONENTS	# BYTES
header (FourCC set to "DX10")	128
extended header (DXGI format set to DXGI_FORMAT_BC3_UNORM)	20
256-by-64 main image	16384
128-by-32 mipmap image	4096
64-by-16 mipmap image	1024
32-by-8 mipmap image	256
16-by-4 mipmap image	64
8-by-2 mipmap image	32
4-by-1 mipmap image	16
2-by-1 mipmap image	16
1-by-1 mipmap image	16

Related topics

P

DDS Cube Map Example

2/22/2020 • 2 minutes to read • Edit Online

For cubic environment maps, one or more faces of a cube are written to the file, using either uncompressed or compressed formats, and all faces must be the same size. Each face can have mipmaps defined, although all faces must have the same number of mipmap levels. If a file contains a cube map, DDSCAPS_COMPLEX, DDSCAPS2_CUBEMAP, and one or more of DSCAPS2_CUBEMAP_POSITIVEX/Y/Z and/or DDSCAPS2_CUBEMAP_NEGATIVEX/Y/Z should be set. The faces are written in the order: positive x, negative x, positive y, negative y, positive z, negative z, with any missing faces omitted. Each face is written with its main image, followed by any mipmap levels.

For example, a 256-by-256 cube map with positive x, negative y, and positive z faces, a pixel format of DXT1, and all mipmap levels would contain the following:

DDS COMPONENTS	# BYTES
header	128
256-by-256 positive x main image	32768
128-by-128 positive x mipmap image	8192
64-by-64 positive x mipmap image	2048
32-by-32 positive x mipmap image	512
16-by-16 positive x mipmap image	128
8-by-8 positive x mipmap image	32
4-by-4 positive x mipmap image	8
2-by-2 positive x mipmap image	8
1-by-1 positive x mipmap image	8
repeat the previous 9 layers for the y mipmap image	43704
repeat the previous 9 layers for the z mipmap image	43704

Starting with DirectX 8, a cube map is stored with all faces defined.

DXGI Cube Maps

Cubic environment maps in Direct3D 10.x and Direct3D 11 are equivalent to a 2D texture array with 6 images, and can be stored in DDS files as such. With Direct3D 10.1 and Direct3D 11, the hardware can also support arrays of cubemaps which are themselves 2D texture arrays with a multiple of 6 images (6, 12, 18, 24, etc.).

For example, here is a 256-by-256 cubemap with mipmap levels stored in a BC6H format as a 2D texture array:

DDS COMPONENTS	# BYTES
header (FourCC of "DX10")	128
extended header (DXGI format set to 95 [DXGI_FORMAT_BC6H_UF16], dimension value of 3 [D3Dxx_RESOURCE_DIMENSION_TEXTURE2D], array size of 6, misc flags of 0x4 [D3Dxx_RESOURCE_MISC_TEXTURECUBE])	20
256-by-256 array entry 0 (positive x) main image	65536
128-by-128 array entry 0 (positive x) mipmap image	16384
64-by-64 array entry 0 (positive x) mipmap image	4096
32-by-32 array entry 0 (positive x) mipmap image	1024
16-by-16 array entry 0 (positive x) mipmap image	256
8-by-8 array entry 0 (positive x) mipmap image	64
4-by-4 array entry 0 (positive x) mipmap image	16
2-by-2 array entry 0 (positive x) mipmap image	16
1-by-1 array entry 0 (positive x) mipmap image	16
repeat the previous 9 layers for array entry 1 (negative x) mipmap image	87408
repeat the previous 9 layers for array entry 2 (positive y) mipmap image	87408
repeat the previous 9 layers for array entry 3 (negative y) mipmap image	87408
repeat the previous 9 layers for array entry 4 (positive z) mipmap image	87408
repeat the previous 9 layers for array entry 5 (negative z) mipmap image	87408

Related topics

P
r
o
g
r
a
m
m

ing Guide for DDS

DDS Volume Texture Example

2/22/2020 • 2 minutes to read • Edit Online

For a volume texture, use the DDSCAPS_COMPLEX, DDSCAPS2_VOLUME, DDSD_DEPTH, flags and set and dwDepth. A volume texture is an extension of a standard texture for Direct3D 9; a volume texture is can be defined with or without mipmaps.

For volumes without mipmaps, each depth slice is written to the file in order. If mipmaps are included, all depth slices for a given mipmap level are written together, with each level containing half as many slices as the previous level with a minimum of 1.

For example, a 64-by-64-by-4 volume map using a pixel format of R8G8B8 (3 bytes per pixel) with all mipmap levels would contain the following:

DDS COMPONENTS	# BYTES
header	128 bytes
64-by-64 slice 1 of 4 main image.	12288 bytes
64-by-64 slice 2 of 4 main image.	12288 bytes
64-by-64 slice 3 of 4 main image.	12288 bytes
64-by-64 slice 4 of 4 main image.	12288 bytes
32-by-32 slice 1 of 2 mipmap image.	3072 bytes
32-by-32 slice 2 of 2 mipmap image.	3072 bytes
16-by-16 slice 1 of 1 mipmap image.	768 bytes
8-by-8 slice 1 of 1 mipmap image.	192 bytes
4-by-4 slice 1 of 1 mipmap image.	48 bytes
2-by-2 slice 1 of 1 mipmap image.	12 bytes
1-by-1 slice 1 of 1 mipmap image.	3 bytes

Note that the smallest mipmap level is only 3 bytes because the bitcount is 24 and there is no added compression at this level.

Support for volume textures was added in DirectX 8.

Related topics

P
r

DDS Bit Flag Values

2/22/2020 • 2 minutes to read • Edit Online

The content for this page has been moved to the appropriate reference page. If you are using Direct3D 10, the flags are used by **DDS_HEADER**.

Related topics

Programming Guide for DDS

Reference for DDS

2/22/2020 • 2 minutes to read • Edit Online

The DDS reference documentation contains the API elements that define the layout of a DDS file. These API elements are defined in the **DDSWithoutD3DX** source files. This sample is located in the `\Samples\C++\Direct3D10\DDSWithoutD3DX\` folder of the DirectX SDK (June 2010).

- DDS_HEADER
- DDS_HEADER_DX10
- DDS_PIXELFORMAT

Related topics

- DDS

DDS_HEADER structure

2/22/2020 • 4 minutes to read • Edit Online

Describes a DDS file header.

Syntax

```
typedef struct {
  DWORD           dwSize;
  DWORD           dwFlags;
  DWORD           dwHeight;
  DWORD           dwWidth;
  DWORD           dwPitchOrLinearSize;
  DWORD           dwDepth;
  DWORD           dwMipMapCount;
  DWORD           dwReserved1[11];
  DDS_PIXELFORMAT ddspf;
  DWORD           dwCaps;
  DWORD           dwCaps2;
  DWORD           dwCaps3;
  DWORD           dwCaps4;
  DWORD           dwReserved2;
} DDS_HEADER;
```

Members

dwSize

Size of structure. This member must be set to 124.

Type: DWORD

dwFlags

Type: DWORD

Flags to indicate which members contain valid data.

FLAG	DESCRIPTION	VALUE
DDSD_CAPS	Required in every .dds file.	0x1
DDSD_HEIGHT	Required in every .dds file.	0x2
DDSD_WIDTH	Required in every .dds file.	0x4

FLAG	DESCRIPTION	VALUE
DDSD_PITCH	Required when pitch is provided for an uncompressed texture.	0x8
DDSD_PIXELFORMAT	Required in every .dds file.	0x1000
DDSD_MIPMAPCOUNT	Required in a mipmapped texture.	0x20000
DDSD_LINEARSIZE	Required when pitch is provided for a compressed texture.	0x80000
DDSD_DEPTH	Required in a depth texture.	0x800000

NOTE

When you write .dds files, you should set the DDSD_CAPS and DDSD_PIXELFORMAT flags, and for mipmapped textures you should also set the DDSD_MIPMAPCOUNT flag. However, when you read a .dds file, you should not rely on the DDSD_CAPS, DDSD_PIXELFORMAT, and DDSD_MIPMAPCOUNT flags being set because some writers of such a file might not set these flags.

The DDS_HEADER_FLAGS_TEXTURE flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSD_CAPS, DDSD_HEIGHT, DDSD_WIDTH, and DDSD_PIXELFORMAT flags.

The DDS_HEADER_FLAGS_MIPMAP flag, which is defined in Dds.h, is equal to the DDSD_MIPMAPCOUNT flag.

The DDS_HEADER_FLAGS_VOLUME flag, which is defined in Dds.h, is equal to the DDSD_DEPTH flag.

The DDS_HEADER_FLAGS_PITCH flag, which is defined in Dds.h, is equal to the DDSD_PITCH flag.

The DDS_HEADER_FLAGS_LINEARSIZE flag, which is defined in Dds.h, is equal to the DDSD_LINEARSIZE flag.

dwHeight

Surface height (in pixels).

Type: DWORD

dwWidth

Surface width (in pixels).

Type: DWORD

dwPi

Type: DWORD

tchOrLinearSize	The pitch or number of bytes per scan line in an uncompressed texture; the total number of bytes in the top level texture for a compressed texture. For information about how to compute the pitch, see the DDS File Layout section of the Programming Guide for DDS.	
dwDepth	Depth of a volume texture (in pixels), otherwise unused.	Type: DWORD
dwMipMapCount	Number of mipmap levels, otherwise unused.	Type: DWORD
dwReserved1	Unused.	Type: DWORD

[11]

ddspf

The pixel format (see DDS_PIXELFORMAT).

Type: DDS_PIXELFORMAT

dwCaps

Type: DWORD

Specifies the complexity of the surfaces stored.

FLAG	DESCRIPTION	VALUE
DDSCAPS_COMPLEX	Optional; must be used on any file that contains more than one surface (a mipmap, a cubic environment map, or mipmapped volume texture).	0x8
DDSCAPS_MIPMAP	Optional; should be used for a mipmap.	0x400000
DDSCAPS_TEXTURE	Required	0x1000

> **NOTE**
>
> When you write .dds files, you should set the DDSCAPS_TEXTURE flag, and for multiple surfaces you should also set the DDSCAPS_COMPLEX flag. However, when you read a .dds file, you should not rely on the DDSCAPS_TEXTURE and DDSCAPS_COMPLEX flags being set because some writers of such a file might not set these flags.

The DDS_SURFACE_FLAGS_MIPMAP flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS_COMPLEX and DDSCAPS_MIPMAP flags.

The DDS_SURFACE_FLAGS_TEXTURE flag, which is defined in Dds.h, is equal to the DDSCAPS_TEXTURE flag.

The DDS_SURFACE_FLAGS_CUBEMAP flag, which is defined in Dds.h, is equal to the DDSCAPS_COMPLEX flag.

dwCaps2

Type: DWORD

Additional detail about the surfaces stored.

FLAG	DESCRIPTION	VALUE
DDSCAPS2_CUBEMAP	Required for a cube map.	0x200
DDSCAPS2_CUBEMAP_POSITIVEX	Required when these surfaces are stored in a cube map.	0x400
DDSCAPS2_CUBEMAP_NEGATIVEX	Required when these surfaces are stored in a cube map.	0x800
DDSCAPS2_CUBEMAP_POSITIVEY	Required when these surfaces are stored in a cube map.	0x1000
DDSCAPS2_CUBEMAP_NEGATIVEY	Required when these surfaces are stored in a cube map.	0x2000
DDSCAPS2_CUBEMAP_POSITIVEZ	Required when these surfaces are stored in a cube map.	0x4000
DDSCAPS2_CUBEMAP_NEGATIVEZ	Required when these surfaces are stored in a cube map.	0x8000
DDSCAPS2_VOLUME	Required for a volume texture.	0x200000

The DDS_CUBEMAP_POSITIVEX flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS2_CUBEMAP and DDSCAPS2_CUBEMAP_POSITIVEX flags.

The DDS_CUBEMAP_NEGATIVEX flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS2_CUBEMAP and DDSCAPS2_CUBEMAP_NEGATIVEX flags.

The DDS_CUBEMAP_POSITIVEY flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS2_CUBEMAP and DDSCAPS2_CUBEMAP_POSITIVEY flags.

The DDS_CUBEMAP_NEGATIVEY flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS2_CUBEMAP and DDSCAPS2_CUBEMAP_NEGATIVEY flags.

The DDS_CUBEMAP_POSITIVEZ flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS2_CUBEMAP and DDSCAPS2_CUBEMAP_POSITIVEZ flags.

The DDS_CUBEMAP_NEGATIVEZ flag, which is defined in Dds.h, is a bitwise-OR combination of the DDSCAPS2_CUBEMAP and DDSCAPS2_CUBEMAP_NEGATIVEZ flags.

The DDS_CUBEMAP_ALLFACES flag, which is defined in Dds.h, is a bitwise-OR combination of the DDS_CUBEMAP_POSITIVEX, DDS_CUBEMAP_NEGATIVEX, DDS_CUBEMAP_POSITIVEY, DDS_CUBEMAP_NEGATIVEY, DDS_CUBEMAP_POSITIVEZ, and DDSCAPS2_CUBEMAP_NEGATIVEZ flags.

The DDS_FLAGS_VOLUME flag, which is defined in Dds.h, is equal to the DDSCAPS2_VOLUME flag.

NOTE

Although Direct3D 9 supports partial cube-maps, Direct3D 10, 10.1, and 11 require that you define all six cube-map faces (that is, you must set DDS_CUBEMAP_ALLFACES).

d w C	Unused.	Type: DWORD

aps3		
dwCaps4	Unused.	Type: DWORD
dwReserved2	Unused.	Type: DWORD

Remarks

Include flags in **dwFlags** for the members of the structure that contain valid data.

Use this structure in combination with a **DDS_HEADER_DXT10** to store a resource array in a DDS file. For more information, see texture arrays.

DDS_HEADER is identical to the DirectDraw DDSURFACEDESC2 structure without DirectDraw dependencies.

Requirements

Header	Dds.h

See also

Reference

DDS_HEADER_DXT10 structure

2/22/2020 • 2 minutes to read • Edit Online

DDS header extension to handle resource arrays, DXGI pixel formats that don't map to the legacy Microsoft DirectDraw pixel format structures, and additional metadata.

Syntax

```
typedef struct {
  DXGI_FORMAT              dxgiFormat;
  D3D10_RESOURCE_DIMENSION resourceDimension;
  UINT                     miscFlag;
  UINT                     arraySize;
  UINT                     miscFlags2;
} DDS_HEADER_DXT10;
```

Members

dxgiFormat	The surface pixel format (see DXGI_FORMAT).	Type: DXGI_FORMAT
resourceDimension		Type: D3D10_RESOURCE_DIMENSION

Identifies the type of resource. The following values for this member are a subset of the values in the D3D10_RESOURCE_DIMENSION or D3D11_RESOURCE_DIMENSION enumeration:

TYPE	DESCRIPTION	VALUE
DDS_DIMENSION_TEXTURE1D (D3D10_RESOURCE_DIMENSION_TEXTURE1D)	Resource is a 1D texture. The **dwWidth** member of **DDS_HEADER** specifies the size of the texture. Typically, you set the **dwHeight** member of **DDS_HEADER** to 1; you also must set the DDSD_HEIGHT flag in the **dwFlags** member of **DDS_HEADER**.	2
DDS_DIMENSION_TEXTURE2D (D3D10_RESOURCE_DIMENSION_TEXTURE2D)	Resource is a 2D texture with an area specified by the **dwWidth** and **dwHeight** members of **DDS_HEADER**. You can also use this type to identify a cube-map texture. For more information about how to identify a cube-map texture, see **miscFlag** and **arraySize** members.	3
DDS_DIMENSION_TEXTURE3D (D3D10_RESOURCE_DIMENSION_TEXTURE3D)	Resource is a 3D texture with a volume specified by the **dwWidth**, **dwHeight**, and **dwDepth** members of **DDS_HEADER**. You also must set the DDSD_DEPTH flag in the **dwFlags** member of **DDS_HEADER**.	4

miscFlag

Type: UINT

Identifies other, less common options for resources. The following value for this member is a subset of the values in the D3D10_RESOURCE_MISC_FLAG or D3D11_RESOURCE_MISC_FLAG enumeration:

TYPE	DESCRIPTION	VALUE
DDS_RESOURCE_MISC_TEXTURECUBE	Indicates a 2D texture is a cube-map texture.	0x4

arraySize

Type: UINT

The number of elements in the array.

For a 2D texture that is also a cube-map texture, this number represents the number of cubes. This number is the same as the number in the **NumCubes** member of D3D10_TEXCUBE_ARRAY_SRV1 or D3D11_TEXCUBE_ARRAY_SRV). In this case, the DDS file contains **arraySize***6 2D textures. For more information about this case, see the **miscFlag** description.

For a 3D texture, you must set this number to 1.

miscFlags2

Type: UINT

Contains additional metadata (formerly was reserved). The lower 3 bits indicate the alpha mode of the associated resource. The upper 29 bits are reserved and are typically 0.

TYPE	DESCRIPTION	VALUE
DDS_ALPHA_MODE_UNKNOWN	Alpha channel content is unknown. This is the value for legacy files, which typically is assumed to be 'straight' alpha.	0x0
DDS_ALPHA_MODE_STRAIGHT	Any alpha channel content is presumed to use straight alpha.	0x1
DDS_ALPHA_MODE_PREMULTIPLIED	Any alpha channel content is using premultiplied alpha. The only legacy file formats that indicate this information are 'DX2' and 'DX4'.	0x2
DDS_ALPHA_MODE_OPAQUE	Any alpha channel content is all set to fully opaque.	0x3
DDS_ALPHA_MODE_CUSTOM	Any alpha channel content is being used as a 4th channel and is not intended to represent transparency (straight or premultiplied).	0x4

NOTE

The legacy D3DX 10 and D3DX 11 utility libraries will fail to load any .DDS file with **miscFlags2** not equal to zero.

Remarks

Use this structure together with a DDS_HEADER to store a resource array in a DDS file. For more info, see texture arrays.

This header is present if the **dwFourCC** member of the DDS_PIXELFORMAT structure is set to 'DX10'.

Requirements

Header	Dds. h

See also

Reference for DDS

DDS_PIXELFORMAT structure

2/22/2020 • 2 minutes to read • Edit Online

Surface pixel format.

Syntax

```
struct DDS_PIXELFORMAT {
  DWORD dwSize;
  DWORD dwFlags;
  DWORD dwFourCC;
  DWORD dwRGBBitCount;
  DWORD dwRBitMask;
  DWORD dwGBitMask;
  DWORD dwBBitMask;
  DWORD dwABitMask;
};
```

Members

dwSize

Structure size; set to 32 (bytes).

Type: DWORD

dwFlags

Values which indicate what type of data is in the surface.

Type: DWORD

FLAG	DESCRIPTION	VALUE
DDPF_ALPHAPIXELS	Texture contains alpha data; **dwRGBAlphaBitMask** contains valid data.	0x1
DDPF_ALPHA	Used in some older DDS files for alpha channel only uncompressed data (dwRGBBitCount contains the alpha channel bitcount; dwABitMask contains valid data)	0x2

FLAG	DESCRIPTION	VALUE
DDPF_FOURCC	Texture contains compressed RGB data; **dwFourCC** contains valid data.	0x4
DDPF_RGB	Texture contains uncompressed RGB data; **dwRGBBitCount** and the RGB masks (**dwRBitMask**, **dwGBitMask**, **dwBBitMask**) contain valid data.	0x40
DDPF_YUV	Used in some older DDS files for YUV uncompressed data (dwRGBBitCount contains the YUV bit count; dwRBitMask contains the Y mask, dwGBitMask contains the U mask, dwBBitMask contains the V mask)	0x200
DDPF_LUMINANCE	Used in some older DDS files for single channel color uncompressed data (dwRGBBitCount contains the luminance channel bit count; dwRBitMask contains the channel mask). Can be combined with DDPF_ALPHAPIXELS for a two channel DDS file.	0x20000

dwFo

Type: DWORD

urCC

Four-character codes for specifying compressed or custom formats. Possible values include: *DXT1*, *DXT2*, *DXT3*, *DXT4*, or *DXT5*. A FourCC of DX10 indicates the prescense of the **DDS_HEADER_DXT10** extended header, and the dxgiFormat member of that structure indicates the true format. When using a four-character code, dwFlags must include *DDPF_FOURCC*.

dwRGBBitCount

Type: DWORD

Number of bits in an RGB (possibly including alpha) format. Valid when **dwFlags** includes *DDPF_RGB*, *DDPF_LUMINANCE*, or *DDPF_YUV*.

dwRBitMask

Type: DWORD

Red (or lumiannce or Y) mask for reading color data. For instance, given the A8R8G8B8 format, the red mask would be 0x00ff0000.

dwGBitMask

Type: DWORD

Green (or U) mask for reading color data. For instance, given the A8R8G8B8 format, the green mask would be 0x0000ff00.

dwBBitMask

Type: DWORD

Blue (or V) mask for reading color data. For instance, given the A8R8G8B8 format, the blue mask would be 0x000000ff.

dwABitMask

Type: DWORD

Alpha mask for reading alpha data. dwFlags must include *DDPF_ALPHAPIXELS* or *DDPF_ALPHA*. For instance, given the A8R8G8B8 format, the alpha mask would be 0xff000000.

Remarks

To store DXGI formats such as floating-point data, use a **dwFlags** of DDPF_FOURCC and set **dwFourCC** to 'D','X','1','0'. Use the DDS_HEADER_DXT10 extension header to store the DXGI format in the **dxgiFormat** member.

Note that there are non-standard variants of DDS files where **dwFlags** has DDPF_FOURCC and the **dwFourCC** value is set directly to a D3DFORMAT or DXGI_FORMAT enumeration value. It is not possible to disambiguate the D3DFORMAT versus DXGI_FORMAT values using this non-standard scheme, so the DX10 extension header is recommended instead.

Requirements

Header	Dds.h

See also

Refe

renceforDDS