INGROUP_BUG

Generated by Doxygen 1.5.9

Mon May 17 18:54:48 2010

Contents

Chapter 1

Shapes API Documentation

1.1 Introduction

[This section should talk briefly about the API itself. Should give a brief overview of what the API enables.] Shapes API provides a framework to manipulate various geometrical shapes like Rectangles,circles etc. It also has the ability to use the underlying GPU to accelerate the processing of the shapes.

1.2 Another Section

Multiple sections can be added using the "section" doxygen tag.

1.2.1 A Sub Section

To better document the introduction, one can also use subsection tags. This will appear under the "Another Section" Section.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Shapes APIs

Modules

- Mandatory APIs
- Configuration APIs
- Operational APIs
- Enumerated Constants
- Data Structures

5.1.1 Detailed Description

This group groups all the APIs in the Shapes API module.

5.2 Mandatory APIs

- uint32 Shapes_Init (void ∗handle)
  Initializes the Shapes API.
- uint32 Shapes_DeInit (void ∗pHandle)
  Deinitializes the Shapes API.

5.2.1 Detailed Description

This is an example group which groups mandatory APIs.

5.2.2 Function Documentation

5.2.2.1 uint32 Shapes_DeInit (void ∗ *pHandle*)

Deinitializes the Shapes API.

This function de-initializes various internal modules that manage the shapes and also powers off the GPU.

Parameters:

↔ ∗***pHandle*** Pointer to the handle structure that will be cleared upon successful de-initialization

Shapes_Init must have been called prior to this function

Any Shapes APIs that were being executed at the time of deinit will have undefined behavior. So care must be taken to ensure all pending operations were completed prior to the de-initialization.

Returns:

Returns one of the following values. The values can be nicely presented using item tags as follows:

- SHAPES_SUCCESS
- SHAPES_ERROR

See also:

Shapes_Init

5.2.2.2 uint32 Shapes_Init (void ∗ *handle*)

Initializes the Shapes API.

This function initializes various internal modules that manage the shapes and also powers on the GPU and restores it to a known default state.

Parameters:

→ ∗***pHandle*** Pointer to the handle structure that will be updated upon successful initialization

None

None

Returns:

Returns one of the following values. The values can be nicely presented using item tags as follows:

- SHAPES_SUCCESS
- SHAPES_ERROR

See also:

Shapes_DeInit

5.3 Configuration APIs

- void Shapes_EnableGPU (bool bEnable)

  Enable/disables the GPU.

- bool Shapes_GetGPUState ()

  Get the state of the GPU.

- void Shapes_SetDbgLvl (Shapes_DbgLvlType eDbgLvl)

  Sets the current debug level.

- Shapes_DbgLvlType Shapes_GetDbgLvl ()

  Gets the current debug level.

5.3.1 Detailed Description

This example section groups all the APIs used to configure the Shapes API

5.3.2 Function Documentation

5.3.2.1 void Shapes_EnableGPU (bool *bEnable*)

Enable/disables the GPU.

Enables or disables the GPU based on the value of bEnable. If bEnable is TRUE GPU is enabled, disabled otherwise.

Parameters:

← ***bEnable*** Boolean to enable the GPU

Shapes_Init must have been called prior to this function

None

Returns:

None

See also:

Shapes_GetGPUState

5.3.2.2 Shapes_DbgLvlType Shapes_GetDbgLvl ()

Gets the current debug level.

Gets the debug level current configured with the Shapes API

Shapes_Init must have been called prior to this function

None

Returns:

returns the debug level current configured with the Shapes API.

See also:

Shapes_SetDbgLvl

5.3.2.3 bool Shapes_GetGPUState ()

Get the state of the GPU.

Get the Graphics Processing Unit's State

Shapes_Init must have been called prior to this function

None

Returns:

Returns TRUE if GPU is enabled; FALSE Otherwise.

See also:

Shapes_EnableGPU

5.3.2.4 void Shapes_SetDbgLvl (Shapes_DbgLvlType *eDbgLvl*)

Sets the current debug level.

Sets the debug level for the Shapes API

Parameters:

← ***eDbgLvl*** Debug level to be used by the Shapes API

Shapes_Init must have been called prior to this function

None

Returns:

None.

See also:

Shapes_GetDbgLvl

5.4 Operational APIs

- void Shapes_DrawPoint (void ∗pHandle, Shapes_PtType mPt)

  Draws a point.

- void Shapes_DrawRect (void ∗pHandle, Shapes_RectType mRect)

  Draws a rectangle.

- void Shapes_GetRectPerimeter (void ∗pHandle, Shapes_RectType mRect, uint32 ∗dwPerimeter)

  Gets the perimeter of the rectangle.

5.4.1 Detailed Description

This example section groups all the APIs that manipulate and draw shapes supported by the Shapes API

5.4.2 Function Documentation

5.4.2.1 void Shapes_DrawPoint (void ∗ *pHandle*, Shapes_PtType *mPt*)

Draws a point.

Draws a point using mPt for its co-ordinates.

Parameters:

← ∗***pHandle*** Pointer to the handle from Shapes_Init

← ***mPt*** Point to be drawn

Shapes_Init must have been called prior to this function

None

Returns:

None

See also:

Shapes_DrawRect

5.4.2.2 void Shapes_DrawRect (void ∗ *pHandle*, Shapes_RectType *mRect*)

Draws a rectangle.

Draws a rectangle using mRect for its co-ordinates.

Parameters:

← ∗***pHandle*** Pointer to the handle from Shapes_Init

← ***mRect*** Rectangle to be drawn

Shapes_Init must have been called prior to this function

None

Returns:

None

See also:

Shapes_DrawPoint

5.4.2.3 void Shapes_GetRectPerimeter (void $*$ *pHandle*, Shapes_RectType *mRect*, uint32 $*$ *dwPerimeter*)

Gets the perimeter of the rectangle.

Computes the perimeter of the given rectangle using the following formula: [Doxygen Allows us to use LaTeX Formula for LaTeX/PDF only outputs] $perimeter = 2 * (length + width)$ where $length = mRect.mTopLeft.x - mRect.mBottomRight.x$ and $width = mRect.mTopLeft.y - mRect.mBottomRight.y$

Parameters:

- $\leftarrow$ $*$***pHandle*** Pointer to the handle from Shapes_Init
- $\leftarrow$ ***mRect*** Rectangle whose perimeter to be computed
- $\rightarrow$ $*$***dwPerimeter*** Pointer to store the perimeter value

Shapes_Init must have been called prior to this function

None

Returns:

None

See also:

None

5.5 Enumerated Constants

- enum Shapes_ShapeType { SHAPE_SQUARE = 0, SHAPE_RECT = 1, SHAPE_CIRCLE = 2, SHAPE_ELLIPSE = 3 }
- enum Shapes_DbgLvlType {
  DBGLVL_ALL = 0, DBGLVL_INFO = 1, DBGLVL_WARN = 2, DBGLVL_ERROR = 3,
  DBGLVL_FATAL = 4 }

5.5.1 Detailed Description

This section defines the various enumerated constants. Please note that all the enums defined here are actually typedef'ed enums and can be used without the explicit enum keyword.

5.5.2 Enumeration Type Documentation

5.5.2.1 enum Shapes_DbgLvlType

Debug Levels used by the Shapes API to display messages.

Enumerator:

DBGLVL_ALL – Everything gets printed

DBGLVL_INFO – Informative messages

DBGLVL_WARN – Warning messages

DBGLVL_ERROR – Non-critical error messages

DBGLVL_FATAL – Critical error messages.

5.5.2.2 enum Shapes_ShapeType

Various shapes supported by the Shapes API

Enumerator:

SHAPE_SQUARE – quadrilateral with four right angles and unequal lengths

SHAPE_RECT – quadrilateral with four right angles and equal lengths

SHAPE_CIRCLE – Curve with equal major and minor radius

SHAPE_ELLIPSE – Curve with unequal major and minor radius

5.6 Data Structures

Data Structures

- struct Shapes_PtType

5.6.1 Detailed Description

This section defines the various data structures published in the Shapes API. Please note that all the structs defined here are actually typedef'ed struct and can be used without the explicit struct keyword.

Chapter 6

Data Structure Documentation

6.1 Shapes_PtType Struct Reference

`#include <Shapes.h>`

Data Fields

- uint32 x
- uint32 y

6.1.1 Detailed Description

A structure that represents a point

6.1.2 Field Documentation

6.1.2.1 uint32 Shapes_PtType::x

– The X coordinate

6.1.2.2 uint32 Shapes_PtType::y

– The Y coordinate

The documentation for this struct was generated from the following file:

- Shapes.h

6.2 Shapes_RectType Struct Reference

`#include <Shapes.h>`

Data Fields

- Shapes_PtType mTopLeft
- Shapes_PtType mBottomRight

6.2.1 Detailed Description

A structure that represents a rectangle

6.2.2 Field Documentation

6.2.2.1 Shapes_PtType Shapes_RectType::mBottomRight

– Point that represents the bottom right corner of the rectangle

6.2.2.2 Shapes_PtType Shapes_RectType::mTopLeft

– Point that represents the top left corner of the rectangle

The documentation for this struct was generated from the following file:

- Shapes.h

Chapter 7

File Documentation

7.1 Shapes.h File Reference

This file contains the definitions of the constants, data structures, and interfaces that comprise the Shapes API. Only the interfaces declared shall be used by the client for accessing the Shapes API.

Data Structures

- struct Shapes_PtType
- struct Shapes_RectType

Defines

Return Codes

The following constants are grouped as Return Codes using the "name" doxygen tag

- #define SHAPES_SUCCESS 0x0
- #define SHAPES_ERROR 0x8001

Enumerations

- enum Shapes_ShapeType { SHAPE_SQUARE = 0, SHAPE_RECT = 1, SHAPE_CIRCLE = 2, SHAPE_ELLIPSE = 3 }
- enum Shapes_DbgLvlType {
  DBGLVL_ALL = 0, DBGLVL_INFO = 1, DBGLVL_WARN = 2, DBGLVL_ERROR = 3,
  DBGLVL_FATAL = 4 }

Functions

- uint32 Shapes_Init (void ∗handle)

  Initializes the Shapes API.

- uint32 Shapes_DeInit (void ∗pHandle)

  Deinitializes the Shapes API.

- void Shapes_EnableGPU (bool bEnable)
  Enable/disables the GPU.
- bool Shapes_GetGPUState ()
  Get the state of the GPU.
- void Shapes_SetDbgLvl (Shapes_DbgLvlType eDbgLvl)
  Sets the current debug level.
- Shapes_DbgLvlType Shapes_GetDbgLvl ()
  Gets the current debug level.

- void Shapes_DrawPoint (void ∗pHandle, Shapes_PtType mPt)
  Draws a point.
- void Shapes_DrawRect (void ∗pHandle, Shapes_RectType mRect)
  Draws a rectangle.
- void Shapes_GetRectPerimeter (void ∗pHandle, Shapes_RectType mRect, uint32 ∗dwPerimeter)
  Gets the perimeter of the rectangle.

7.1.1 Detailed Description

This file contains the definitions of the constants, data structures, and interfaces that comprise the Shapes API. Only the interfaces declared shall be used by the client for accessing the Shapes API.

Anything that starts after a blank line is a detailed description. This is also a sample header file that serves as a template as a sample documentation for doxygen. Make sure you replace the DDIShapes.h above with the name of the header file

7.1.2 Define Documentation

7.1.2.1 #define SHAPES_ERROR 0x8001

– The API Failed for some reason

7.1.2.2 #define SHAPES_SUCCESS 0x0

– The API Succeeded

Index