Pluggable Window System

API

Filename	:	Pluggable Window System.API.1.7.doc
Version	:	1.0.449a External Issue
Issue Date	:	13 Jun 2011
Author	:	PowerVR

Contents

Function API Listing

1. Introduction

1.1. Scope

This document provides implementation guidelines for a loadable Window System EGL (WSEGL) module that provides the necessary interface between the PowerVR OpenGLES driver and the native window system. These guidelines include usage of the PVR2D functions that must be used if the customer is controlling the allocation of the drawable buffer memory pages.

The module will be responsible for such things as creating window drawables, enumerating display capabilities and providing any surface information required by OpenGLES to perform a render.

1.2. Intended Audience

The audience of this document should be familiar with the following APIs and systems:

- Khronos EGL
- OpenGLES
- Windows System such as Series 60, the X11 Window system, JavaVM, or WinCE Window System

1.3. Related Documents

PowerVR 2D.PVR2D API.doc

Khronos OpenGLES specification. http://www.khronos.org/opengles/spec.html

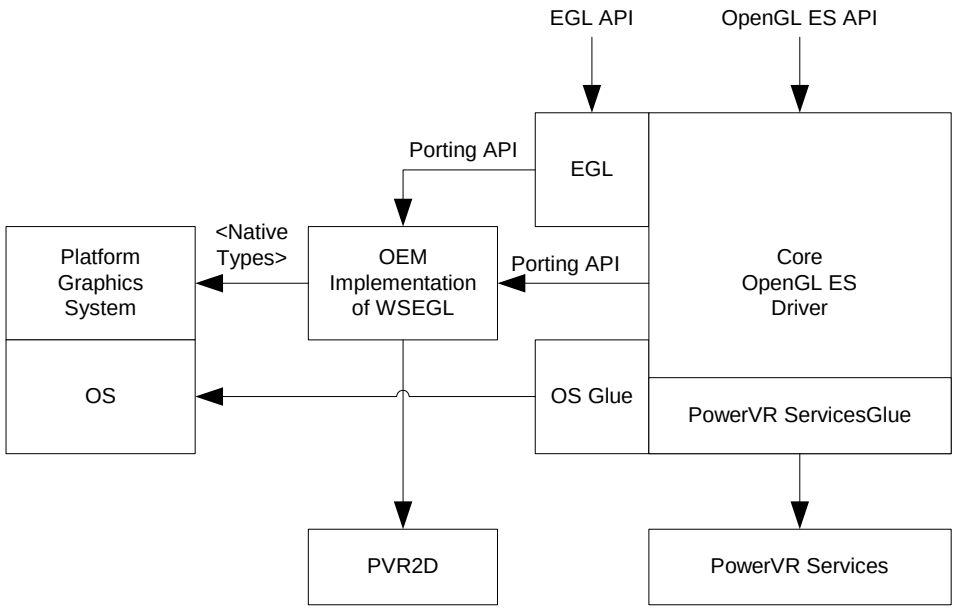
Khronos EGL specification. http://www.khronos.org/opengles/spec.html

1.4. Terminology

Term	Meaning
OpenGL®-ES™	The 3D drawing API that is defined by the Khronos Group, and which is being facilitated through this document. OpenGL is a registered trademark of Silicon Graphics Inc.
Application	The software that calls the egl*() and gl*() entry points
WSEGL library	A module that implements the APIs described in this document
OEM implementation	An OEMs particular implementation of the WSEGL library

2. System Description

2.1. Architecture/System diagram



OGL/EGL/WS/Services Interaction

2.2. Introduction

The WSEGL is a set of Porting API's that the OEM implements to provide an interface between the OpenGL ES Driver and the platform Native Types. The Native Types may be specific to the platform and not defined by an external body.

This document describes the interfaces between the OEM Implementation of the WSEGL as indicated in the above diagram by the arrowed line labeled Porting API.

2.3. Native Types

Most window systems provide for the concept of a Display, a Window and an off-screen image or Pixmap. The Khronos EGL spec was designed to allow these objects to be typedef'd as the NativeTypes required for the EGL API. To implement this API the OEM implementation must create appropriate typedefs for the Khronos EGL Native Types, these are:

- NativeDisplayType
- NativePixmapType
- NativeWindowType

The types that are defined must fit into the storage provided by a 'C' void * as these are passed through the EGL API in the OpenGL ES driver. That means that in general these types must themselves be pointers, or integers less than or equal to the storage size of a pointer.

The NativeTypes are transparent to the OpenGL-ES driver, and are passed through to the OEM implementation. The OEM implementation should use its own methods to extract the information it needs from these NativeTypes, for instance the OEM implementation will need to provide a function to retrieve the dimensions of the pixmap or window through its own method.

2.4. Loading WSEGL Library

As mentioned above, the WSEGL library is a dynamically loaded library. The WSEGL library is loaded by the EGL component of the OpenGLES driver, when the first eglGetDisplay() call is received. The code is loaded by following a set of rules to determine which WSEGL library is loaded. e.g.:

- Per native application settings stored in a file – determined by process name, or system wide default
- Via names and heuristics that are hard-coded into the OpenGLES driver (EG. If the X Server is running then use libpvrX11WSEGL.so)

In most cases the WSEGL library that OEMs have implemented will be loaded using the setting stored in a file as noted in the first method above.

Note: If all the APIs described in this document are not present in the WSEGL, then the OpenGL-ES driver will fail to load the module.

2.5. OpenGL ES and WSEGL Functions

While some WSEGL functions have implied counterparts to OpenGL ES EGL functions, the OEM implementation should not presume that there will be a 1 to 1 correspondence between egl*() and gl*() functions and WSEGL_*() functions.

2.6. HW Prerequisites

Hardware rendering to a surface is restricted to a set of pixel formats, with certain conditions on the stride and address of that surface.

2.6.1. Surface Pixel formats for rendering:

The pixel format of the surface must be one of the following:

- RGB 565

Bits	31:27	26:21	20:16	15:11	10:5	4:0
Component	R1	G1	B1	R0	G0	B0

- ARGB 4444

Bits	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
Component	A1	R1	G1	B1	A0	R0	G0	B0

- ARGB 1555

Bits	31	30:26	25:21	20:16	15	14:10	9:5	4:0
Component	A1	R1	G1	B1	A0	R0	G0	B0

- ARGB 8888

Bits	31:24	23:16	15:8	7:0
Component	A0	R0	G0	B0

- ABGR 8888

Bits	31:24	23:16	15:8	7:0
Component	A0	B0	G0	R0

- XBGR 8888

Bits	31:24	23:16	15:8	7:0
Component	Unused	B0	G0	R0

2.6.2. Y'C_BC_R pixel formats

Y'C_BC_R pixel formats can be used only with external EGL images. It is not possible to use Y'C_BC_R pixel formats with renderable surfaces.

The exact colour space encoding for Y'C_BC_R pixel formats is controlled with the following flags:

Flag	Meaning
WSEGL_FLAGS_YUV_CONFORMANT_RANGE	Y' data is in the range [16,235] and C_BC_R data is in the range [16,239]
WSEGL_FLAGS_YUV_FULL_RANGE	Y' data is in the range [0,255] and C_BC_R data is in the range [0,255]
WSEGL_FLAGS_YUV_BT601	Y'C_BC_R colour space conforms to ITU-R Recommendation BT.601
WSEGL_FLAGS_YUV_BT709	Y'C_BC_R colour space conforms to ITU-R Recommendation BT.709

The colour space encoding is selected by setting suitable flags of the ulFlags field in WSEGLDrawableParams structures that are retrieved by WSEGL_GetDrawableParameters().

The memory layouts of supported Y'C_BC_R pixel formats are listed below. The ui32Stride in WSEGLDrawableParams is defined as the Y' plane stride.

WSEGL_PIXELFORMAT_NV12

This is a planar Y'C_BC_R pixel format with two planes and 4:2:0 chroma subsampling.

- Plane 1: 8 bit per pixel Y' plane
- Followed immediately by Plane 2: C_BC_R plane with 16 bit word encoding a 2x2 block of pixels
- Equivalent to http://www.fourcc.org/yuv.php#NV12
- Width: 0 to 2048
- Y' Plane Stride: (Width + 31) & ~31
- C_BC_R Plane Stride: ((Width/2) + 31) & ~31

Plane 1: Packed Y' data

Bits	31:24	23:16	15:8	7:0
Component	Y'3	Y'2	Y'1	Y'0

Plane 2: Packed C_BC_R data

Bits	31:24	23:16	15:8	7:0
Component	C_R1	C_B1	C_R0	C_B0

WSEGL_PIXELFORMAT_I420

This is a planar Y'C_BC_R pixel format with three planes and 4:2:0 chroma subsampling.

- Plane 1: 8 bit per pixel Y' plane
- Followed immediately by Plane 2: C_B plane with 8 bit byte encoding a 2x2 block of pixels
- Followed immediately by Plane 3: C_R plane with 8 bit byte encoding a 2x2 block of pixels
- Equivalent to http://www.fourcc.org/yuv.php#NV12
- Width: 0 to 2048
- Y' Plane Stride: (Width + 31) & ~31
- C_BC_R Plane Stride: ((Width/2) + 31) & ~31

Plane 1: Packed Y' data

Bits	31:24	23:16	15:8	7:0
Component	Y'3	Y'2	Y'1	Y'0

Plane 2: Packed C_B data

Bits	31:24	23:16	15:8	7:0
Component	C_B3	C_B2	C_B1	C_B0

Plane 3: Packed C_R data

Bits	31:24	23:16	15:8	7:0
Component	C_R3	C_R2	C_R1	C_R0

WSEGL_PIXELFORMAT_YUYV

This is a non-planar packed $Y'C_BC_R$ pixel format with 4:2:2 chroma subsampling.

- 32 bit word encoding a 2x1 block of pixels

Bits	31:24	23:16	15:8	7:0
Component	C_R0	Y'1	C_B0	Y'0

2.6.3. Stride

The stride of image data must be a multiple of 32 pixels.

2.6.4. Alignment

The base address of image data must be 4 byte aligned.

2.6.5. SGX MMU mapping

The image data must fit within the available SGX MMU slots. The MMU has enough slots to cover 256MB of memory and enough range to map memory over 4GB. Each MMU slot maps a 4k byte page, which is aligned to the platform's 4k page. As the MMU is also used to map texture and parameter memory, not all the slots are available to map render target surfaces / YUV video.

This image data will be mapped in to the MMU via one or several function calls to PVR2D. These calls can either allocate and wrap a piece of memory, or just wrap a previously allocated piece of memory; when wrapping the memory it can be either contiguous or not.

To allow memory to be wrapped the OEM implementation must provide the physical page address of the underlying memory, and ensure that the virtual to physical mapping has been locked.

3. APIs

The following are the porting API's that the OEM is required to implement.
These are subject to change.

3.1. Types

```
#define WSEGL_FALSE 0
#define WSEGL_TRUE 1

/*
// WSEGL handles
*/
typedef void *WSEGLDisplayHandle;
typedef void *WSEGLDrawableHandle;

/*
// Display capability type
*/
typedef enum WSEGLCapsType_TAG
{
        WSEGL_NO_CAPS = 0,
        WSEGL_CAP_MIN_SWAP_INTERVAL = 1, /* System default value = 1 */
        WSEGL_CAP_MAX_SWAP_INTERVAL = 2, /* System default value = 1 */
        WSEGL_CAP_WINDOWS_USE_HW_SYNC = 3, /* System default value = 0 (FALSE) */
        WSEGL_CAP_PIXMAPS_USE_HW_SYNC = 4, /* System default value = 0 (FALSE) */
} WSEGLCapsType;

/*
// Display capability
*/
typedef struct WSEGLCaps_TAG
{
        WSEGLCapsType eCapsType;
        unsigned long ui32CapsValue;

} WSEGLCaps;

/*
// Drawable type
*/
#define WSEGL_NO_DRAWABLE             0x0
#define WSEGL_DRAWABLE_WINDOW         0x1
#define WSEGL_DRAWABLE_PIXMAP         0x2


/*
// Pixel format of display/drawable
*/
typedef enum WSEGLPixelFormat_TAG
{
        WSEGL_PIXELFORMAT_RGB565 = 0,
        WSEGL_PIXELFORMAT_ARGB4444 = 1,
        WSEGL_PIXELFORMAT_ARGB8888 = 2,
        WSEGL_PIXELFORMAT_ARGB1555 = 3,
        WSEGL_PIXELFORMAT_ABGR8888 = 4

} WSEGLPixelFormat;

/*
// Transparent of display/drawable
*/
typedef enum WSEGLTransparentType_TAG
{
        WSEGL_OPAQUE = 0,
        WSEGL_COLOR_KEY = 1,

} WSEGLTransparentType;


/*
// Display/drawable configuration
*/
typedef struct WSEGLConfig_TAG
{
        /* Type of drawables this configuration applies to -
           OR'd values of drawable types.
        */
        unsigned long ulDrawableType;

        /* Pixel format */
```

```
	WSEGLPixelFormat ePixelFormat;

	/* Native Renderable  - set to WSEGL_TRUE if native renderable */
	unsigned long ulNativeRenderable;

	/* FrameBuffer Level Parameter */
	unsigned long ulFrameBufferLevel;

	/* Native Visual ID */
	unsigned long ulNativeVisualID;

	/* Native Visual */
	void *hNativeVisual;

	/* Transparent Type */
	WSEGLTransparentType eTransparentType;

	/* Transparent Color – only used if transpaerent type is COLOR_KEY */
	unsigned long ulTransparentColor; /* packed as 0x00RRGGBB */

} WSEGLConfig;

/*
// WSEGL errors
*/
typedef enum WSEGLError_TAG
{
	WSEGL_SUCCESS = 0,
	WSEGL_CANNOT_INITIALISE = 1,
	WSEGL_BAD_NATIVE_DISPLAY = 2,
	WSEGL_BAD_NATIVE_WINDOW = 3,
	WSEGL_BAD_NATIVE_PIXMAP = 4,
	WSEGL_BAD_NATIVE_ENGINE = 5,
	WSEGL_BAD_DRAWABLE = 6,
	WSEGL_BAD_MATCH = 7,
	WSEGL_OUT_OF_MEMORY = 8

} WSEGLError;

/*
// Drawable orientation (in degrees anti-clockwise)
*/
typedef enum WSEGLRotationAngle_TAG
{
	WSEGL_ROTATE_0 = 0,
	WSEGL_ROTATE_90 = 1,
	WSEGL_ROTATE_180 = 2,
	WSEGL_ROTATE_270 = 3

} WSEGLRotationAngle;

/*
// Drawable information required by OpenGL-ES driver
*/
typedef struct WSEGLDrawableParams_TAG
{
	/* Width in pixels of the drawable */
	unsigned long  ui32Width;

	/* Height in pixels of the drawable */
	unsigned long  ui32Height;

	/* Stride in pixels of drawable */
	unsigned long  ui32Stride;

	/* Pixel format of the drawable */
	WSEGLPixelFormat      ePixelFormat;

	/* User space cpu virtual address of drawable */
	void           *pvLinearAddress;

	/* HW address of drawable */
	unsigned long  ui32HWAddress;

	/* Private data for the drawable */
	void                  *hPrivateData;
```

```
} WSEGLDrawableParams;
```

3.2. Function Listing

Mandatory functions. All the APIs described in this list must be present in the WSEGL but some may be empty, depending on the implementation.

- WSEGL_IsDisplayValid
- WSEGL_InitialiseDisplay
- WSEGL_CloseDisplay
- WSEGL_CreateWindowDrawable
- WSEGL_CreatePixmapDrawable
- WSEGL_DeleteDrawable
- WSEGL_SwapDrawable
- WSEGL_SwapControlInterval
- WSEGL_WaitNative
- WSEGL_CopyFromDrawable
- WSEGL_CopyFromPBuffer
- WSEGL_GetDrawableParameters
- WSEGL_ConnectDrawable
- WSEGL_DisconnectDrawable

Implementation-specific functions. These functions only exist on the implementations specified, but still may be empty.

- WSEGL_SetDisplayProperty (Symbian OS)

WSEGL_IsDisplayValid

```
WSEGLError WSEGL_IsDisplayValid (
               NativeDisplayType          hDisplay,
               );
```

Inputs

hDisplay	Native display, or 0 for default display

Outputs

Returns

WSEGL_SUCCESS	Success
WSEGL_BAD_NATIVE_DISPLAY	hDisplay doesn't match a display

Description

This function checks that a display in the window system matching hDisplay is available, where hDisplay describes a specific display provided by the system that the OpenGLES driver is running on. A value of 0 for hDisplay checks for the existence of a default display. If more than one display exists, the OEM implementation is free to select which display corresponds to the native display, but must select one.

WSEGL_InitialiseDisplay

```
WSEGLError WSEGL_InitialiseDisplay (
               NativeDisplayType          hNativeDisplay,
               WSEGLDisplayHandle*        phDisplay,
               WSEGLCaps**                psCapabilities,
               WSEGLConfigs**             psConfigs
               );
```

Inputs

hNativeDisplay	The native display, or 0 for the default native display

Outputs

psConfigs	The configs available on the specified display
psCapabilities	The capabilities of the specified display
phDisplay	The WSEGLDisplay handle for the specified display

Returns

WSEGL_SUCCESS	Success
WSEGL_OUT_OF_MEMORY	Memory was not available to initialise the display
WSEGL_BAD_MATCH	No configs are available on this display.
WSEGL_BAD_NATIVE_DISPLAY	hNativeDisplay doesn't correspond to a valid native display
WSEGL_CANNOT_INITIALISE	The display is valid but cannot be initialised for some reason

Description

This function passes the NativeDisplayType parameter passed into the eglGetDisplay function to the OEM implementation. The OEM implementation should initialise the specified display and return a unique handle to the display which will be used to further reference the display in later calls to the OEM implementation through the porting API.

This function must return the configs available on this display. The config array must be terminated by the last array entry's ulDrawableType field containing the value WSEGL_NO_DRAWABLE. If no configs are available for this display, then the function should fail and return WSEGL_BAD_MATCH. Each config in the array may support both window and pixmap drawables. The drawable types supported are indicated by OR'ing the values into the ulDrawableType field. However, all other values in the config are singular, and must be correct for all the drawable types supported by the config.

In addition the function must return the capabilities of the display in a WSEGLCaps struct array. The caps array must be terminated by the last array entries eCapsType field containing the value WSEGL_NO_CAP. If no capabilities or are specified, a stucture array of dimension 1 and containing the terminator value should be returned.

The lifespan of both these arrays is until WSEGL_CloseDisplay is called with the WSEGLDisplayHandle returned by this call.

The OEM implementation must reference count successful calls to WSEGL_InitialiseDisplay to allow multiple calls (potentially from different threads) to initialise the same display. Multiple calls should not create multiple instances of the display or the capabilities and config arrays. The returned handle from multiple calls to initialise the same display must be identical.

If 0 is passed in as the hNativeDisplay, and more than one display exists, the OEM implementation is free to select which display corresponds to the native display, but must select one.

WSEGL_CloseDisplay

```
WSEGLError WSEGL_CloseDisplay (
              WSEGLDisplayHandle         hDisplay
              );
```

Inputs

hDisplay	The display to close

Outputs

Returns

WSEGL_SUCCESS	Success

Description

This function closes a previously initialised display. The function should decrement the reference count for this display and only close the display when the reference count reaches zero. At this point the capabilities structure array returned from the initialise call can deleted, and the unique display handle be made available for reuse.

A display that has been closed must allow further the calls to re-initialise the display.

WSEGL_CreateWindowDrawable

```
WSEGLError WSEGL_CreateWindowDrawable (
                WSEGLDisplayHandle          hDisplay,
                WSEGLConfig*                psConfig,
                WSEGLDrawableHandle*        phDrawable,
                NativeWindowType            hNativeWindow,
                WSEGLRotationAngle*         eRotationAngle
                );
```

Inputs

`hDisplay`	The display on which to create a drawable
`psConfig`	The configuration to use
`hNativeWindow`	The native window

Outputs

`phDrawable`	The newly created window drawable handle
eRotationAngle	The rotation angle of the window

Returns

`WSEGL_SUCCESS`	Success
`WSEGL_BAD_MATCH`	psConfig doesn't correspond to a valid config for hNativeWindow
`WSEGL_BAD_NATIVE_WINDOW`	hNativeWindow doesn't correspond to a valid native window
`WSEGL_OUT_OF_MEMORY`	Memory was not available to create the drawable

Description

Creates a window drawable for a native window using the specified configuration and returns a unique handle. This handle will be used to further reference the drawable in later calls, until the drawable is deleted by a call to WSEGL_DeleteDrawable.

The native window must be validated against the configuration. The configuration will be one of the window supporting configurations returned by InitialiseDisplay. If the window doesn't match the config then the error WSEGL_BAD_MATCH must be returned. As windows are backbuffered, if the visible surface of the window doesn't match that of the config, but the OEM implementation can support the color transform on swap, then the config may still be validated, but the drawing buffer created in this call must match the config.

The OEM implementation is responsible for obtaining the memory to be used for the colour buffer associated with the window, and also for obtaining the relevant memory addresses needed in the `WSGLES_GetDrawableParameters` call. There are two methods for doing this:

1. Allow PVR2d to allocate and map the memory to the SGX MMU slots, by using the PVR2DMemAlloc* functions.

2. The function obtains the memory itself and then uses the PVR2DMemWrap* functions to get the memory mapped into the SGX MMU. This approach requires that the OEM implementation has the physical page address for the memory in question, and has locked the memory in place.

WSEGL_CreatePixmapDrawable

```
WSEGLError WSEGL_CreatePixmapDrawable (
               WSEGLDisplayHandle          hDisplay,
               WSEGLConfig*                psConfig,
               WSEGLDrawableHandle*        phDrawable,
               NativePixmapType            hNativePixmap,
               WSEGLRotationAngle*         eRotationAngle
               );
```

Inputs

hDisplay	The display on which to create a drawable
psConfig	The configuration to use
hNativePixmap	The native pixmap

Outputs

phDrawable	The newly created pixmap drawable handle
eRotationAngle	The rotation angle of the pixmap

Returns

WSEGL_SUCCESS	Success
WSEGL_BAD_MATCH	hNativePixmap doesn't correspond to the config.
WSEGL_BAD_NATIVE_PIXMAP	hNativePixmap doesn't correspond to a valid native pixmap
WSEGL_OUT_OF_MEMORY	Memory was not available to create the drawable

Description

Creates a pixmap drawable for a native window using the specified configuration and returns a unique handle. This handle will be used to further reference the drawable in later calls, until the drawable is deleted by a call to WSEGL_DeleteDrawable.

The native pixmap must be validated against the configuration. The configuration will be one of the pixmap supporting configurations returned by InitialiseDisplay. If the pixmap doesn't match the config then the error WSEGL_BAD_MATCH must be returned.

The pixmap memory must be mapped to SGX MMU slots. To do this there are two possible methods:

1. Ensure that the pixel memory associated with the Pixmap was originally allocated by PVR2D This ensures the memory is already mapped to the SGX MMU. This would be done by using the PVR2DMemAlloc* functions.
2. The OEM implementation derives the phyical page addresses for the pixel memory associated with the Pixmap, and then uses the PVR2DMemWrap* functions to get the memory mapped into the SGX MMU. The memory must be locked in place before calling the PVR2D functions.

WSEGL_DeleteDrawable

```
WSEGLError WSEGL_DeleteDrawable (
               WSEGLDrawableHandle        hDrawable
               );
```

Inputs

hDrawable	The drawable to delete

Outputs

Returns

WSEGL_SUCCESS	Success

Description

Deletes a previously created window, or pixmap drawable, created by WSEGL_CreateWindow, or WSEGL_CreatePixmapDrawable. Any resources associated with the drawable, such as colour buffers or wrapped memory, can be deleted and its unique handle is available for reuse.

The OpenGL-ES driver is responsible for ensuring that all rendering activity has finished on this surface before this function is called.

WSEGL_SwapDrawable

```
WSEGLError WSEGL_SwapDrawable (
               WSEGLDrawableHandle        hDrawable,
               unsigned long              ui32Data
               );
```

Inputs

`hDrawable`	The drawable to swap
`ui32Data`	Single integer value, see below.

Outputs

Returns

`WSEGL_SUCCESS`	Success
`WSEGL_BAD_DRAWABLE`	The swap cannot be carried out for an implementation dependant reason
`WSEGL_BAD_NATIVE_WINDOW`	The Native Window associated with this drawable is no longer valid.

Description

This function copies the colour buffer of a window drawable to the native window associated with that drawable. It is defined in the EGL specification that the colour buffer of the drawable is left in an undefined state, and it is possible for the WSEGL implementer to ensure that this is defined in a particular way.

This function will only be called for window drawables

The data parameter to the call will by default contain an incrementing count of the number of calls to eglSwapDrawable that have occurred on this drawable. The parameter may contain other values on the basis of EGL extensions.

WSEGL_SwapControlInterval

```
WSEGLError WSEGL_SwapControlInterval (
               WSEGLDrawablehandle        hDrawable,
               unsigned long              ui32Interval
               );
```

Inputs

`hDrawable`	The drawable whose swap interval is to bet set
`ui32Interval`	The swap interval

Outputs

Returns

`WSEGL_SUCCESS`	Success

Description

This function sets the swap interval of a window drawable, the swap interval being the minimum number of video frames that are displayed before the posting of colour buffer data to the native window can occur. The value passed into this function will be clamped to the minimum and maximum values indicated by the display capabilities. If these capabilities are not present, then the value will be clamped to a maximum of one and a minimum of one. If these capabilities are not present this function can be implemented as an empty function.

See the EGL 1.1 specification for more details on how swap interval is intended to operate.

WSEGL_WaitNative

```
WSEGLError WSEGL_WaitNative (
               WSEGLDrawableHandle        hDrawable,
               unsigned long              ui32NativeEngine
               );
```

Inputs

`hDrawable`	The drawable to flush
`ui32NativeEngine`	The engine to wait for

Outputs

Returns

`WSEGL_SUCCESS`	Success
`WSEGL_BAD_NATIVE_ENGINE`	ui32NativeEngine doesn't correspond to a known marking engine

Description

This function flushes native rendering requests on a drawable. Any rendering by the specified marking engine (another drawing API) is guaranteed to have completed by the time that this function returns. In addition in the OEM implementation is using performance enhancing methods such as changing caching on HW renderable surfaces, then these should be disabled in this call.

While this call is the counterpart to `eglWaitNative()` it should not be assumed that it is only called in response to `eglWaitNative()`.

The NativeEngine parameter is a platform dependant value defined by the OEM implementation as per NativeWindowType etc. If an unknown NativeEngine is supplied the error WSEGL_BAD_NATIVE_ENGINE should be returned. A NativeEngine value of zero indicates that the OEM implementation should wait for all non-3D rendering engines that are being employed to complete on the drawable in question.

WSEGL_CopyFromDrawable

```
WSEGLError WSEGL_CopyFromDrawable (
               WSEGLDrawableHandle        hDrawable,
               NativePixmaptype           hNativePixmap
               );
```

Inputs

hDrawable	Source drawable
hNativePixmap	Target native pixmap

Outputs

Returns

WSEGL_SUCCESS	Success
WSEGL_BAD_NATIVE_PIXMAP	The Native Pixmap associated with this drawable is no longer valid.
WSEGL_BAD_NATIVE_PIXMAP	The target native pixmap is invalid
WSEGL_BAD_NATIVE_WINDOW	The Native Window associated with this drawable is no longer valid.
WSEGL_BAD_MATCH	The OEM implementation does not support the color conversion necessary to copy the drawable data to the Native Pixmap

Description

Copies the colour buffer of a drawable to a native pixmap, leaving the colour buffer of the source drawable unchanged.

If the pixel format of the drawable does not match the format of the Native Pixmap the OEM implementation may perform the conversion or fail the copy and return the error WSEGL_BAD_MATCH.

WSEGL_CopyFromPBuffer

```
WSEGLError WSEGL_CopyFromPBuffer (
               void*                    pvAddress,
               unsigned long            ui32Width,
               unsigned long            ui32Height,
               unsigned long            ui32Stride,
               WSEGLPixelFormat         ePixelFormat,
               NativePixmapType         hNativePixmap
               );
```

Inputs

`pvAddress`	Address of the source PBuffer surface
`ui32Width`	Width (in pixels) of the PBuffer
`ui32Height`	Height (in pixels) of the PBuffer
`ui32Stride`	Stride (in pixels) of the PBuffer
`ePixelFormat`	Pixel format of the PBuffer
`hNativePixmap`	Target native pixmap

Outputs

Returns

`WSEGL_SUCCESS`	Success
`WSEGL_BAD_MATCH`	The OEM implementation does not support the color converision necessary to copy the Pbuffer data to the Native Pixmap
`WSEGL_BAD_NATIVE_PIXMAP`	The target native pixmap is invalid

Description

Copies the colour buffer data of a PBuffer to a native pixmap, leaving the colour buffer of the source PBuffer unchanged. The address parameter `pvAddress`, is a pointer to a linear address that the window system can read through directly.

If the pixel format of the Pbuffer does not match the format of the Native Pixmap the OEM implementation may either perform a conversions or fail the copy and return the error WSEGL_BAD_MATCH.

Note that there is no create function for pbuffers as they are defined to be created by the OpenGL ES driver internally.

WSEGL_GetDrawableParameters

```
WSEGLError WSGLES_GetDrawableParameters (
               WSEGLDrawableHandle        hDrawable,
               WSEGLDrawableParams*       psSourceParams
               WSEGLDrawableParams*       psRenderParams
               );
```

Inputs

`hDrawable`	The psDrawable to retrieve the render parameters for

Outputs

`psSourceParams`	The parameters of the render source
`psRenderParams`	The parameters of the render target

Returns

`WSEGL_SUCCESS`	Success
`WSEGL_BAD_NATIVE_PIXMAP`	The Native Pixmap associated with this drawable is no longer valid.
`WSEGL_BAD_NATIVE_WINDOW`	The Native Window associated with this drawable is no longer valid.
`WSEGL_BAD_DRAWABLE`	The information needed to fill out the structures cannot be obtained.

Description

This function is called by the OpenGLES driver to retrieve the information needed to perform a 3d render, or to perform a glReadPixels() operation, on the specified drawable.

The source drawable parameters `WSEGLDrawableParams` structure specify the source data for the background of a 3d render, whereas the render drawable parameters `WSEGLDrawableParams` structure specify the destination of a 3d render. This allows for background data to be used efficiently rather than it being copied into the destination before the render. These structures may be returned containing the same data.

The HW addresses and private data for synchronisation can be obtained from PVR2D either by allocating the drawable buffer directly through PVR2D, or by using PVR2D to wrap a previously allocated piece of memory. In this case the hPrivateData field of the DrawableParams structure should be filled with the hPrivateData found in the PVR2D MemInfo structure, else this field should be set to NULL.

Values placed in the `WSEGLDrawableParams` structure must be constrained by the HW prerequisites, else no render will occur.

The exact parameters that must be returned by GetDrawableParameters may be affected by the surface's connection status.See WSEGL_ConnectDrawable and WSEGL_DisconnectDrawable for more information.

WSEGL_ConnectDrawable

```
WSEGLError WSGLES_ConnectDrawable (
               WSEGLDrawableHandle        hDrawable
               );
```

Inputs

hDrawable	The psDrawable being connected

Returns

WSEGL_SUCCESS	Success
WSEGL_BAD_DRAWABLE	The platform failed to connect this drawable

Description

This function is called by EGL to indicate that a WEGL drawable is now current, as a read or draw surface (but not necessarily both).

Connection implies that the drawable must be allocated or mapped, and may be used in a render or glReadPixels() operation in the near future.

After a drawable has been connected, more parameters must be returned by GetDrawableParameters().

Specifically, it is necessary to specify pvLinearAddress, ui32HWAddress, hPrivateData and ui32Stride for the given drawable.

WSEGL_DisconnectDrawable

```
WSEGLError WSGLES_DisconnectDrawable (
               WSEGLDrawableHandle        hDrawable
               );
```

Inputs

hDrawable	The psDrawable being disconnected

Returns

WSEGL_SUCCESS	Success
WSEGL_BAD_DRAWABLE	The platform failed to disconnect this drawable

Description

This function is called by EGL to indicate that a WEGL drawable is no longer current, as a read or draw surface (but not necessarily both).

The WSEGL module is expected to reference count connections, to know when a drawable is no longer current as both a read and draw surface in any context.

If the surface is no longer current as a draw and read surface for any context, a reduced number of parameters may be returned by GetDrawableParameters.

Specifically, it is no longer necessary to specify pvLinearAddress, ui32HWAddress, hPrivateData or ui32Stride for the given drawable.

WSEGL_SetDisplayProperty

```
WSEGLError WSEGL_SetDisplayProperty (
               WSEGLDrawableHandle        hDrawable,
               void                       *pvDisplayProperty
               );
```

Inputs

`hDrawable`	The psDrawable to set the display property on
`pvDisplayProperty`	Pointer to DisplayProperty information. This is actually a pointer to a Symbian OS CEGLDisplayProperty object and should be cast back accordingly.

Outputs

Returns

`WSEGL_SUCCESS`	Success
`WSEGL_OUT_OF_MEMORY`	Unable to set the display property

Description

An implementation of the functionality described in the document *OpenGL ES EGLDisplayProperty Architecture.doc*. This document, along with the source of the `egldisplayproperty` library, can be found in the Symbian devkit tree at

…Symbian…\src\COMMON\GENERIC\graphics\OpenGLESDisplayProperty

The properties of the display that may be set are:

Display Clipping Region. The visibility of the window is described in this list of rectangles and the method used to present the buffers must be updated. Note that a change of display rectangle will always involve such a change of clipping regions.

Display Rectangle. This may alter the size and/or position of the window on screen. A change of size may result in the color buffers being destroyed and recreated. A change of position may allow the existing buffers to be used but the presentation method must be updated.

Display Visibility Flag. An implementation may wish to disable all presentation of color buffers if the visibility is disabled. Note that rendering to the buffers will still continue.