Services

3rd Party Display API

Filename	:	Services.3rd Party Display API.doc
Version	:	4.0.122 External Issue
Issue Date	:	04 Mar 2011
Author	:	PowerVR

Contents

1. Terminology

The following terminology is used in this document:

DisplayClass Component	DisplayClass component of the 3rd party display driver
OS Component	OS component of the 3rd party display driver
DDK	Driver Development Kit. A software package containing driver source code, allowing a specific driver to be built\modified for use on a specific platform
PVR	PowerVR
UMA	Unified Memory Architecture – graphics device addresses system memory either allocated from the OS or reserved in the system memory
LMA	Local Memory Architecture – graphics device has its own block of memory, separate from system memory.

2. Introduction

2.1. Scope

This document specifies the interfaces for 3rd party display class device integration into 'version 4' of Services. An overview of device class integration services component design is also provided. In addition, a 'use case' example is presented to illustrate:

- API usage
- Functionality required by 3rd party devices for services integration

2.2. Driver Integration Guidelines

The 3rd party driver is a specification of an API to integrate the PowerVR Services driver with 3rd Party display hardware. It is NOT a specification for a display controller driver, rather a specification to extend the API for a pre-existing driver for the display hardware.

The 3rd party driver interface provides PowerVR client drivers (e.g. PVR2D) with an API abstraction of the system's underlying display hardware, allowing the client drivers to indirectly control the display hardware and access its associated memory.

Supplied DDK code is intended to be an example of how a pre-existing display driver may be extended to support the 3rd Party Display interface to PowerVR Services – Imagination Technologies is not providing a display driver implementation.

When porting Services to a new system and/or OS it is expected that a driver will already exist for the display hardware, e.g. FBDEV driver, Linux. The interface functions specified in this document should ideally be implemented by extending the interface functions of the existing display hardware driver.

2.3. Hardware Architecture of Display Controllers

Consumer electronic devices incorporate varying classes of display controllers.

- 'Conventional' display controllers update an LCD/CRT at the refresh rate of the display.
- 'Single Shot' displays update LCD panels dynamically rather than at the display refresh rate and therefore improve power usage by minimising bus transactions.
- 'Multi-channel dynamic update' display controllers construct the display's output from one of more independent 'channels', each responsible for a sub-region on the display.
- 'Overlay' based devices provide 'planes' with which the pixels on the display are composed.

These devices may also support any combination of:

- Pixel format conversion
- Colour space conversion
- Scaling
- Inter channel/plane blending

The interfaces described in this document attempt to abstract the underlying architecture of specific display hardware and provide a common control interface to varying classes of display devices.

2.4. Related Documents

Services Software Architectural Specification
Services Software Functional Specification
SGX DDK Porting Guide for Services 4.0

2.5. Overview of Display Class Architecture

Design considerations:

- Important for independent display hardware to be 'coordinated' with services devices
- 3rd party display API provides a consistent interface between services and 3rd Party display device drivers
- Abstracts control of display hardware via the Display Class API (used by OGLES, D3DM, DDraw, windowing system driver, etc.)
- Provides maximum performance while maintaining the order of operations on shared resources.

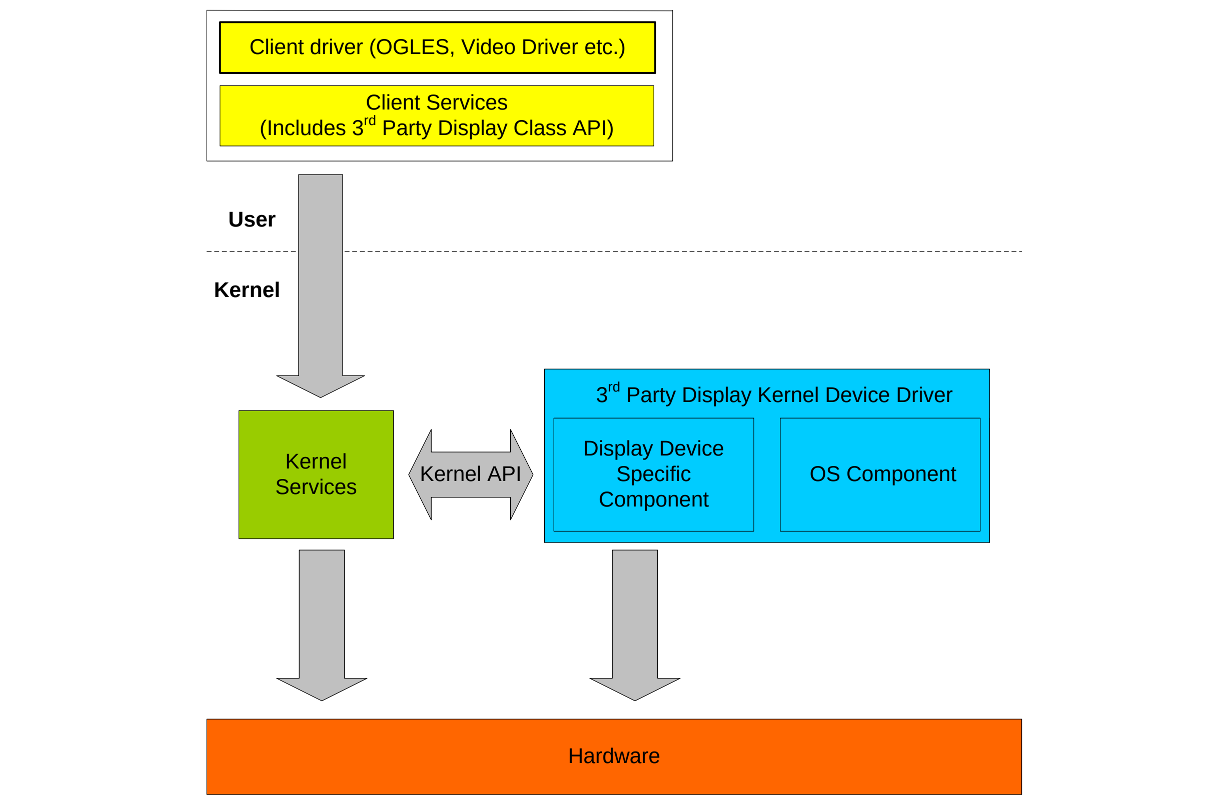

Figure 1 Display Class Architecture

Figure 1 highlights three distinct software components:

1. **Client Driver**: This directly or indirectly interfaces with a client application. It has a 'Client Services' component built into it which provides the services API and the 3rd party Display Class API.

2. **Kernel Services**: The 'kernel mode' Services component

3. **3rd Party Display Kernel Device Driver**: This is a kernel device driver for controlling the display hardware. 'Interfacing code' is added to the driver allowing the display device to be integrated with other Services managed devices. There are two sub components:

    - **Display Device Specific Component:** This contains display device specific code to implement callback functions from Kernel Services.
    - **OS Component:** This is component provides OS abstraction functions

The Arrows in the diagram represent calls through interfaces between software and hardware components. The labelled arrow, 'Kernel API' is described in the following sections.

2.6. SGX and Display Addressable Surfaces

The 3rd Party display class API specification provides support for direct access to displayable surfaces by SGX devices. SGX devices can read to and/or write from display addressable surfaces therefore the display surfaces attributes are limited by the constraints of both the display and SGX devices.

The primary difference between many display devices and SGX is the surface stride granularity, where SGX is designed with a stride granularity of 8 pixels although some core variants are limited to 16 or 32 pixels.

3. The 3rd Party Private Device Data Structure

The 3rd party private device data structure is usually called XXX_DEVINFO, with the 'XXX' being a reference to the 3rd party display hardware. XXX_DEVINFO is allocated and initialized by the 3rd party driver's 'Init' function.

Within the 3rd party driver, a global pointer is used to access XXX_DEVINFO (i.e. a static variable declared within the DisplayClass component).

OpenDCDevice (a services-to-display API) returns an XXX_DEVINFO handle to kernel services. Kernel Services retains a copy of this handle and uses it as an argument to other services-to-display APIs.

Note: XXX_DEVINFO can only be accessed within the 3rd party driver. Here is an example XXX_DEVINFO definition:

```
typedef struct XXX_DEVINFO_TAG
{
        IMG_UINT32                    ui32DeviceID;
        DISPLAY_INFO                  sDisplayInfo;

        /* system surface info */
        XXX_BUFFER                    sSystemBuffer;
        DISPLAY_FORMAT                sSysFormat;
        DISPLAY_DIMS                  sSysDims;

        /* number of supported display formats */
        IMG_UINT32                    ui32NumFormats;

        /* list of supported display formats */
        DISPLAY_FORMAT                asDisplayFormatList[XXX_MAXFORMATS];

        /* number of supported display dims */
        IMG_UINT32                    ui32NumDims;

        /* list of supported display formats */
        DISPLAY_DIMS                  asDisplayDimList[XXX_MAXDIMS];

        /* jump table into PVR services */
        PVRSRV_DC_DISP2SRV_KMJTABLE   sPVRJTable;

        /* jump table into DC */
        PVRSRV_DC_SRV2DISP_KMJTABLE   sDCJTable;

        /* handle for connection to kernel services - OS specific - may not be required */
        IMG_HANDLE                    hPVRServices;

        /* back buffer info */
        XXX_BUFFER                    asBackBuffers[XXX_MAX_BACKBUFFERS];
        DISPLAY_FORMAT                sBackBufferFormat[XXX_MAXFORMATS];

        /* set of vsync flip items - enough for 1 outstanding flip per back buffer */
        PVRPDP_VSYNC_FLIP_ITEM        asVSyncFlips[PVRPDP_MAX_BACKBUFFERS];
        IMG_UINT32                    ui32InsertIndex;
        IMG_UINT32                    ui32RemoveIndex;

        XXX_FBINFO                    sFBInfo;       /* fb info structure */
        IMG_UINT32                    ui32RefCount;  /* ref count */
        XXX_SWAPCHAIN                 *psSwapChain;

}  XXX_DEVINFO;
```

4. The 3rd Party Kernel Driver initialisation

This section describes the 3rd party kernel driver initialisation and de-initialisation functions. The API is not fixed and the specific implementation is the decision of the 3rd party driver writer. However, the descriptions serve as a template for how to structure the initialisation of a 3rd party kernel driver.

Initialisation is expected to occur at 3rd party kernel driver load time. It is important that the 3rd party driver is loaded after kernel services.

Init

```
PVRSRV_ERROR Init();
```

Inputs

Outputs

Returns

PVRSRV_ERROR_OUT_OF_MEMORY	Memory allocation failure
PVRSRV_ERROR_INIT_FAILURE	Initialisation failure
PVRSRV_ERROR_DEVICE_REGISTER_FAILED	Failed to register device with services
PVRSRV_OK	Success

Component

Implemented in the DisplayClass component of the 3rd party kernel driver

Description

- Allocates and sets up the device's private data structure (i.e. XXX_DEVINFO)
- Initiates a connection to kernel services
- Acquires the kernel services Display Class jump table, enabling calls from the 3rd party device into kernel services.
- Sets up its own Display Class jump table, enabling calls from kernel services into the 3rd party driver.
- Registers the device as a display class device type with kernel services
- Registers the device's private flip/swap command handler with kernel services command queue manager.
- Installs a Vsync ISR and associated handler function
- A pointer to the XXX_DEVINFO, the device's private data structure, is stored in a static variable.

Deinit

```
PVRSRV_ERROR Deinit();
```

Inputs

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the DisplayClass component of the 3rd party kernel driver

Description

This function is called when the 3rd party kernel driver is unloaded. It decrements the reference count and will perform the following de-initialisation tasks if the reference count is equal to zero:

- Uninstall VSYNC ISR
- Remove private command processing functions from the Command Queue Manager
- Remove the device from the services display device class list, de-allocating any associated resources
- Closes the connection to kernel services
- De-allocates any resources in the 3rd party kernel driver.

5. 'OS Component' Functions

This section outlines some of the functions that should be implemented in the OS component of the 3rd party display driver. Note: the API is not fixed and the specific implementation is the decision of the 3rd party driver writer.

OpenPVRServices

```
PVRSRV_ERROR OpenPVRServices (IMG_HANDLE *phPVRServices);
```

Outputs

phPVRServices	Handle for connection from 3rd party kernel driver to kernel services

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_INVALID_DEVICE`	Fail

Component

Implemented in the OS component of the 3rd party kernel driver

Description

This function opens a connection from a 3rd party kernel driver to the kernel services. This must be called before any other kernel APIs in this section.

Note: depending on the Operating System, this API may or may not be required

ClosePVRServices

```
PVRSRV_ERROR ClosePVRServices (IMG_HANDLE hPVRServices);
```

Inputs

hPVRServices	Handle for connection from 3rd party kernel driver to kernel services

Outputs

Returns

`PVRSRV_OK`	Success

Component

Implemented in the OS component of the 3rd party kernel driver

Description

This function closes a connection from a 3rd party kernel driver to the kernel services. It is implemented in the OS component 3rd party kernel driver.

Note: depending on the Operating System, this API may or may not be required

6. Kernel API

The 'Kernel API' arrow (shown in Figure 1) describes the connection between kernel services and the 3rd party display driver. More specifically, the arrow represents two interfaces each flowing in opposing directions:

1. Display-to-Services: APIs called from within the 3rd party display driver and implemented in kernel services, see 'Display to Services' Kernel API.
2. Services-to-Display: APIs called from within kernel services and implemented in the 3rd party display driver, see 'Services to Display' Kernel API.

6.1. 'Display to Services' Kernel API

The DisplayClass component contains the 'consumer services knowledge' that must be built into the 3rd party kernel driver in order to interface with kernel services directly.

The APIs described here are retrieved from kernel services via a data structure of function pointers. Kernel services exports the function **PVRGetDisplayClassJTable** which the 3rd party display driver must call in order to acquire the function pointers.

PVRGetDisplayClassJTable

```
IMG_EXPORT IMG_BOOL PVRGetDisplayClassJTable(PVRSRV_DC_DISP2SRV_KMJTABLE
                   *psJTable);
```

Inputs

psJTable	Structure of function pointers each corresponding to a kernel services API function

Outputs

Returns

IMG_TRUE	Success
IMG_FALSE	Fail

Component

Implemented in kernel services

Description

This function is used by the 3rd party display driver to retrieve kernel services API functions via a pre-defined structure of function pointers. Each API is documented in the following sections.

For clarity the function pointer structure definition is also presented:

```
/* Function table for DISPLAY->SRVKM */
typedef struct PVRSRV_DC_DISP2SRV_KMJTABLE_TAG
{
        IMG_UINT32                          ui32TableSize;
        PFN_DC_REGISTER_DISPLAY_DEV         pfnPVRSRVRegisterDCDevice;
        PFN_DC_REMOVE_DISPLAY_DEV           pfnPVRSRVRemoveDCDevice;
        PFN_DC_OEM_FUNCTION                 pfnPVRSRVOEMFunction;
        PFN_DC_REGISTER_COMMANDPROCLIST     pfnPVRSRVRegisterCmdProcList;
        PFN_DC_REMOVE_COMMANDPROCLIST       pfnPVRSRVRemoveCmdProcList;
        PFN_DC_CMD_COMPLETE                 pfnPVRSRVCmdComplete;
        PFN_DC_REGISTER_SYS_ISR             pfnPVRSRVRegisterSystemISRHandler;
        PFN_DC_REGISTER_POWER               pfnPVRSRVRegisterPowerDevice;
} PVRSRV_DC_DISP2SRV_KMJTABLE, *PPVRSRV_DC_DISP2SRV_KMJTABLE;
```

Note: The 3rd party display driver must first acquire a pointer to PVRGetDisplayClassJTable by calling into the OS component (different OS environments will do this in different ways).

PVRSRVRegisterDCDeviceKM

```
PVRSRV_ERROR PVRSRVRegisterDCDeviceKM (PVRSRV_DC_SRV2DISP_KMJTABLE
              *psFuncTable, IMG_UINT32 *pui32DeviceID );
```

Inputs

psFuncTable	Function table for Srvkm->Display

Outputs

pui32DeviceID	Unique identifier index allocated to the 3rd party device

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_OUT_OF_MEMORY	Memory allocation failure
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in kernel services – eurasia\services\srvkm\common\deviceclass.c

Access by display-to-services function pointer

pfnPVRSRVRegisterDCDevice

Description

This function registers the 3rd party device with kernel services (the 3rd party device registers its ‘services-to-display’ function table with kernel services). A device node is allocated and added to the device node list.

The device node has the following attributes and data:

- Device Type: PVRSRV_DEVICE_TYPE_EXT
- Device Class: PVRSRV_DEVICE_CLASS_DISPLAY
- A reference count
- A ‘srvkm-to-display’ jump-table. This mechanism permits 3rd party display driver functionality to be invoked by kernel services. Thus the 3rd party device can be controlled via the display class APIs within client services.

PVRSRVRemoveDCDeviceKM

```
PVRSRV_ERROR PVRSRVRemoveDCDeviceKM(IMG_UINT32 ui32DevIndex);
```

Inputs

ui32DevIndex	Unique device identifier representing the device to remove from consumer services control.

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in kernel services – eurasia\services\srvkm\common\deviceclass.c

Access by display-to-services function pointer

pfnPVRSRVRemoveDCDevice

Description

This function removes ‘control’ of a specified 3rd party device from kernel services. ‘Device Removal’ entails:

- Using the device index supplied by the caller to locate the device node to be removed
- Deleting the device from the device class linked-list so that it cannot be controlled via the display class APIs.
- De-allocating all data structures allocated by PVRSRVRegisterDCDeviceKM, including the ‘srvkm-to-display’ jump-table.
- De-allocating ‘synchronisation object’ temporary storage

PVRSRVOEMFunction

```
PVRSRV_ERROR SysOEMFunction (IMG_UINT32 ui32ID, IMG_VOID *pvIn, IMG_UINT32
              ulInSize, IMG_VOID *pvOut, IMG_UINT32 ulOutSize);
```

Inputs	
ui32ID	Unique ID defining a specific OEM function to be called
pvIn	Data structure associated with OEM function indexed by ui32ID
ulInSize	Size of pvIn
pvOut	Data structure associated with OEM function indexed by ui32ID
ulOutSize	Size of pvOut

Outputs	
pvOut	Data structure associated with OEM function defined by ui32ID

Returns	
`PVRSRV_OK`	Success
`PVRSRV_FAIL`	Fail

Component

Implemented in kernel services – in the eurasia\services\system folder

Access by display-to-services function pointer

pfnPVRSRVOEMFunction

Description

This function is optionally implemented in the system layer kernel services and provides a mechanism for OEMs to customise their software with minimal impact on common kernel services code.

PVRSRVRegisterCmdProcListKM

```
IMG_EXPORT PVRSRV_ERROR PVRSRVRegisterCmdProcListKM (IMG_UINT32
                ui32DevIndex, PFN_CMD_PROC *ppfnCmdProcList, IMG_UINT32
                ui32MaxSyncsPerCmd[][2], IMG_UINT32 ui32CmdCount);
```

Inputs

ui32DevIndex	Device identifier
ppfnCmdProcList	List of private command handler function pointers
ui32MaxSyncsPerCmd	Max number of sync objects used by command
ui32CmdCount	Number of command types the device supports

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_INVALID_PARAMS`	Fail

Component

Implemented in kernel services – eurasia\services\srvkm\common\queue.c

Access by display-to-services function pointer

pfnPVRSRVRegisterCmdProcList

Description

This function registers a list of private command handler function pointers with the command queue manager. The command queue manager allocates temporary storage for each command type based on SRC/DST synchronisation object usage (Note: storage is not allocated for any private command details). Private command handlers are retrieved at command execution time using the device identifier and a command id (an index into the list of private command handlers).

PVRSRVRemoveCmdProcListKM

```
IMG_EXPORT PVRSRV_ERROR PVRSRVRemoveCmdProcListKM (IMG_UINT32 ui32DevIndex,
               IMG_UINT32 ui32CmdCount);
```

Inputs

ui32DevIndex	Device identifier
ui32CmdCount	Number of command types the device supports

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_FAIL`	Fail

Component

Implemented in kernel services – eurasia\services\srvkm\common\queue.c

Access by display-to-services function pointer

pfnPVRSRVRemoveCmdProcList

Description

This function removes a list of private command handler function pointers from the command queue manager. The command queue manager de-allocates temporary storage for each command type.

PVRSRVCommandCompleteKM

```
IMG_EXPORT IMG_VOID PVRSRVCommandCompleteKM(IMG_HANDLE hCmdCookie, IMG_BOOL
               bScheduleMISR);
```

Inputs

hCmdCookie	A pointer to COMMAND_COMPLETE_DATA
bScheduleMISR	Boolean to control if MISR is scheduled

Outputs

Returns

Component

Implemented in kernel services – eurasia\services\srvkm\common\queue.c

Access by display-to-services function pointer

pfnPVRSRVCmdComplete

Description

The 3rd party display driver calls this function after completing a Flip command. The hCmdCookie handle provides access to a COMMAND_COMPLETE_DATA structure, enabling command complete sync objects to be updated.

Note: see Services Functional Specification for details on what an MISR represents

Note: bScheduleMISR should only be passed as TRUE when pfnPVRSRVCmdComplete is called from the display VSync ISR function and when the ISR is a true 'device specific ISR', i.e. when PVRSRVRegisterSystemISRHandler is not used.

PVRSRVRegisterSystemISRHandler

```
PVRSRV_ERROR PVRSRVRegisterSystemISRHandler (PFN_ISR_HANDLER pfnISRHandler,
               IMG_VOID *pvISRHandlerData,
               IMG_UINT32 ui32ISRSourceMask,
               IMG_UINT32 ui32DeviceID);
```

Inputs

pfnISRHandler	Pointer to the device's ISR handling function
pvISRHandlerData	Pointer to the device's ISR handler data
ui32ISRSourceMask	Bit representing the interrupt source device that corresponds to the display device (SOC specific). Frameworks supports a maximum of 32 ganged interrupts.
ui32DeviceID	Device index received at registration and used to associate the ISR handler with the already registered device

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in kernel services – eurasia\services\srvkm\common\deviceclass.c

Access by display-to-services function pointer

pfnPVRSRVRegisterSystemISRHandler

Description

Registers an ISR handling function with kernel services in order to handle interrupts originating from the display device.

Note: This API is only required in the case of 'ganged' SOC interrupt hardware. More specifically, SOCs with a single interrupt line but split into multiple interrupt sources corresponding to discrete devices within the SOC. In this case there is a single top level ISR handling function from which device specific handling functions may be conditionally called based on the actual source of the interrupt.

PVRSRVRegisterPowerDevice

```
PVRSRV_ERROR PVRSRVRegisterPowerDevice (IMG_UINT32 ui32DeviceIndex,
      PFN_PRE_POWER pfnPrePower,
      PFN_POST_POWER pfnPostPower,
      PFN_PRE_CLOCKSPEED_CHANGE     pfnPreClockSpeedChange,
      PFN_POST_CLOCKSPEED_CHANGE    pfnPostClockSpeedChange,
      IMG_HANDLE hDevCookie,
      PVR_POWER_STATE eCurrentPowerState,
      PVR_POWER_STATE eDefaultPowerState);
```

Inputs

ui32DeviceIndex	Device index assigned to the display by services
pfnPrePower	Pointer to call-back function to be called before all services derived power state transitions
pfnPostPower	Pointer to call-back function to be called after all services derived power state transitions
pfnPreClockSpeedChange	Pre clock speed change callback
pfnPostClockSpeedChange	Post clock speed change callback
hDevCookie	Device cookie to be passed to power call-backs
eCurrentPowerState	Current power state
eDefaultPowerState	Default power state

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in kernel services – eurasia\services\srvkm\common\deviceclass.c

Access by display-to-services function pointer

pfnPVRSRVRegisterPowerDevice

Description

This function registers the display device with services power manager. The display device power callbacks will then be called before and after all services derived power transitions

Note: It is generally expected that the display device interfaces directly with the OS power manager and does not register with the services power manager, i.e. this function should not be called unless the display driver cannot integrate with the OS power manager.

6.2. 'Services to Display' Kernel API

These APIs are implemented in the 3rd party display driver and are accessed as follows:

- The client driver (e.g. OPENGLES, Video Driver, etc) calls Client Services Display Class APIs
- Client Services Display Class APIs are routed through to Kernel Services via the client/kernel glue layer
- Kernel Services will invoke DisplayClass APIs by using the 'services-to-display' jump-table – through which 3rd party display HW can be controlled.

The 3rd party display driver uses PVRSRVRegisterDCDeviceKM to pass the jump-table data to kernel services. Here is the jump-table structure:

```
/* Function table for SRVKM->DISPLAY */
typedef struct PVRSRV_DC_SRV2DISP_KMJTABLE_TAG
{
        IMG_UINT32                                  ui32TableSize;
        PFN_OPEN_DC_DEVICE                          pfnOpenDCDevice;
        PFN_CLOSE_DC_DEVICE                         pfnCloseDCDevice;
        PFN_ENUM_DC_FORMATS                         pfnEnumDCFormats;
        PFN_ENUM_DC_DIMS                            pfnEnumDCDims;
        PFN_GET_DC_SYSTEMBUFFER                     pfnGetDCSystemBuffer;
        PFN_GET_DC_INFO                             pfnGetDCInfo;
        PFN_GET_BUFFER_ADDR                         pfnGetBufferAddr;
        PFN_CREATE_DC_SWAPCHAIN                     pfnCreateDCSwapChain;
        PFN_DESTROY_DC_SWAPCHAIN                    pfnDestroyDCSwapChain;
        PFN_SET_DC_DSTRECT                          pfnSetDCDstRect;
        PFN_SET_DC_SRCRECT                          pfnSetDCSrcRect;
        PFN_SET_DC_DSTCK                            pfnSetDCDstColourKey;
        PFN_SET_DC_SRCCK                            pfnSetDCSrcColourKey;
        PFN_GET_DC_BUFFERS                          pfnGetDCBuffers;
        PFN_SWAP_TO_DC_BUFFER                       pfnSwapToDCBuffer;
        PFN_SWAP_TO_DC_SYSTEM                       pfnSwapToDCSystem;
        PFN_SET_DC_STATE                            pfnSetDCState;

} PVRSRV_DC_SRV2DISP_KMJTABLE;
```

OpenDCDevice

```
static PVRSRV_ERROR OpenDCDevice(IMG_UINT32 ui32DeviceID, IMG_HANDLE
                              *phDevice, PVRSRV_SYNC_DATA* psSystemBufferSyncData);
```

Inputs	
ui32DeviceID	Device index assigned by services
psSystemBufferSyncData	System surface sync data

Outputs	
phDevice	Handle to device info structure (XXX_DEVINFO)

Returns	
PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver – XXX_displayclass.c

Access by services-to-display function pointer

pfnOpenDCDevice

Description

Stores the system surface sync data in the device's private data structure (XXX_DEVINFO) and returns a handle to XXX_DEVINFO.

CloseDCDevice

```
static PVRSRV_ERROR CloseDCDevice(IMG_HANDLE hDevice);
```

Inputs

`hDevice`	Handle to device's private data structure

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver – XXX_displayclass.c

Access by services-to-display function pointer

pfnCloseDCDevice

Description

Close connection to the display class device.

EnumDCFormats

```
static PVRSRV_ERROR EnumDCFormats(IMG_HANDLE hDevice, IMG_UINT32
               *pui32NumFormats, DISPLAY_FORMAT *psFormat);
```

Inputs	
hDevice	Handle to 3rd party private data – XXX_DEVINFO

Outputs	
pui32NumFormats	Number of display formats (i.e. pixel format types) supported by the 3rd party display device
psFormat	Pointer to an array of supported display formats

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_INVALID_PARAMS`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnEnumDCFormats

Description

This function returns the number of display formats (i.e. pixel format types) supported by the 3rd party display device, and returns an array of the supported formats (DISPLAY_FORMAT).

This function must be called twice. First, pass NULL for psFormat to return the number of formats via pui32NumFormats. Second, (having allocated space for the format list) psFormat will receive a list of format information structures supported by the display device.

The first entry in the list should correspond to the current mode of the display device (i.e. the mode of the primary surface). When creating a swapchain (via call to CreateSwapChain in the 3rd party display driver), if the pixel format of the swapchain does not match the primary surface, CreateSwapChain must perform a mode-switch (via SetMode). NOTE: the resulting swap chain in this case cannot include the system (windowing system) surface.

EnumDCDims

```
static PVRSRV_ERROR EnumDCDims(IMG_HANDLE hDevice, DISPLAY_FORMAT
                *psFormat, IMG_UINT32 *pui32NumDims, DISPLAY_DIMS *psDim);
```

Inputs	
hDevice	Handle to 3rd party private data – XXX_DEVINFO

Outputs	
pui32NumDims	Number of display dimensions supported by the 3rd party display device
psDim	Array of DISPLAY_DIMS structures

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_INVALID_PARAMS`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnEnumDCDims

Description

This function returns the number of display dimensions (width, height and stride in bytes) supported by the 3rd party display hardware, and returns an array of the supported dimensions (DISPLAY_DIMS).

This function must be called twice. First, pass NULL for psDim to return the number of supported dimensions via pui32NumDims. Second, (having allocated space for the dimension list) psDim receives a list of dimension information structures (DISPLAY_DIMS) supported by the display device.

The first entry in the list should correspond to the current mode of the display device (i.e. the mode of the primary surface). When creating a swapchain (via call to CreateSwapChain in the 3rd party display driver), if the pixel format of the swapchain does not match the primary surface, CreateSwapChain must perform a mode-switch (via SetMode). NOTE: the resulting swap chain in this case cannot include the system (windowing system) surface.

GetDCSystemBuffer

```
static PVRSRV_ERROR GetDCSystemBuffer(IMG_HANDLE hDevice, IMG_HANDLE
                *phBuffer);
```

Inputs	
hDevice	Handle to 3rd party private data – XXX_DEVINFO

Outputs	
phBuffer	Handle to system buffer (primary surface)

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnGetDCSystemBuffer

Description

This function returns a handle to the system buffer (primary surface).

GetDCInfo

```
static PVRSRV_ERROR GetDCInfo(IMG_HANDLE hDevice, DISPLAY_INFO *psDCInfo);
```

Inputs	
hDevice	Handle to 3rd party private data – XXX_DEVINFO

Outputs	
psDCInfo	Pointer to DISPLAY_INFO, which is defined in servicesext.h

Returns	
PVRSRV_OK	Success
PVRSRV_ERROR_INVALID_PARAMS	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnGetDCInfo

Description

This function returns a pointer to the 3rd party display driver's DISPLAY_INFO structure, which is contained within the private XXX_DEVINFO structure.

Note: DISPLAY_INFO is defined in eurasia\include\servicesext.h:

```
/* Display info structure definition */
typedef struct DISPLAY_INFO_TAG
{
        IMG_UINT32      ui32MaxSwapChains;
        IMG_UINT32      ui32MaxSwapChainBuffers;
        IMG_UINT32      ui32MinSwapInterval;
        IMG_UINT32      ui32MaxSwapInterval;
        IMG_CHAR        szDisplayName[MAX_DISPLAY_NAME_SIZE];

#if defined(SUPPORT_HW_CURSOR)
        IMG_UINT16      ui32CursorWidth;
        IMG_UINT16      ui32CursorHeight;
#endif

} DISPLAY_INFO;
```

GetDCBufferAddr

```
static PVRSRV_ERROR GetDCBufferAddr(IMG_HANDLE hDevice, IMG_HANDLE
               hBuffer, IMG_SYS_PHYADDR **ppsSysAddr, IMG_UINT32
               *pui32ByteSize, IMG_VOID **ppvCpuVAddr, IMG_HANDLE
               *phOSMapInfo, IMG_BOOL *pbIsContiguous, IMG_UINT32
               *pui32TilingStride);
```

Inputs	
hDevice	Handle to 3rd party private data – XXX_DEVINFO
hBuffer	Handle to system buffer (primary surface) structure – XXX_BUFFER

Outputs	
ppsSysAddr	System physical address of system buffer
pui32ByteSize	Size of system buffer in bytes
ppvCpuVAddr	CPU virtual address of system buffer
phOSMapInfo	(optional) an OS Mapping handle for KM->UM surface mapping
pbIsContiguous	Indicates whether system is buffer made up of contiguous pages
pui32TilingStride	Tiled stride of the surface if display supports and enables memory tiling on this surface

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_INVALID_PARAMS`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnGetBufferAddr

Description

This function provides the memory mapping information for the system buffer (primary surface). The caller must supply a handle to the data structure containing the system buffer information (i.e. a handle to XXX_BUFFER). The following information is returned:

- The system physical address of the system buffer
- CPU virtual addresses of the system buffer
- The size of the system buffer in bytes
- A boolean value to indicate whether the system buffer memory consists of contiguous or non-contiguous pages.
- OS Mapping handle for KM->UM surface mapping – this is optional and should be set to NULL if unused.

CreateDCSwapChain

```
static PVRSRV_ERROR CreateDCSwapChain(IMG_HANDLE hDevice, IMG_UINT32
               ui32Flags, DISPLAY_SURF_ATTRIBUTES *psDstSurfAttrib,
               DISPLAY_SURF_ATTRIBUTES *psSrcSurfAttrib, IMG_UINT32
               ui32BufferCount, PVRSRV_SYNC_DATA **ppsSyncData, IMG_UINT32
               ui32OEMFlags, IMG_HANDLE *phSwapChain, IMG_UINT32
               *pui32SwapChainID);
```

Inputs	
hDevice	Handle to 3rd party private data – XXX_DEVINFO
ui32Flags	Swap chain control flags
ui32BufferCount	Number of buffers required in this swap chain
ui32OEMFlags	OEM specific flags
psDstSurfAttrib	Destination surface attributes (pixel format, stride, width, height, etc)
psSrcSurfAttrib	Source surface attributes (pixel format, stride, width, height, etc)

Outputs	
phSwapChain	Handle to the newly created swap chain
pui32SwapChainID	Optional swap chain ID

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnCreateDCSwapChain

Description

This function creates a swap chain on the specified device. Depending on the system, the buffers for the swap chain may be:

- Allocated by the 3rd party driver
- Allocated elsewhere, in which case the 3rd party driver is responsible for accessing the buffers and inserting them into the swap chain

If the dimensions and format do not match current mode of the display device, an internal display 'Set Mode' call should be made as part of CreateDCSwapChain and the resulting swap-chain may not include the system (windowing system) surface.

A command queue should be created for each swap chain.

A swap chain consists of one or more backbuffers specified by (psSrcSurfAttrib):

- Height, width, stride, address

- Pixel format
- Selection rectangle (see SetDCSrcRect)

The selection rectangle refers to a rectangular region of the backbuffer that should be presented to the front buffer.

The front buffer has similar attributes:

- Height, width, stride
- Pixel format
- Selection rectangle (pixels from the back buffer selection rectangle are 'presented' to this region) (see SetDCDstRect).

What the front buffer actually represents varies according to the architecture of the underlying display hardware. In the case of conventional display hardware DRAM memory may be allocated for the pixels of the front buffer. However, in other display hardware designs no memory may be allocated and in this case the persistent pixel storage in an LCD panel may represent the front buffer.

Whatever the underlying architecture of the display hardware, it is the responsibility of the private 3rd party drivers to manage the swap chain appropriately.

A command queue should also be created for each swap chain.

Note: the SRC and DST selection rectangles are initialised according to the nature of the display hardware.

The client driver will receive a copy of the swap-chain handle, and must supply it to the client/kernel glue layer when using all other swap-chain related APIs.

Associated structure – located in eurasia\include\servicesext.h:

```
typedef struct DISPLAY_SURF_ATTRIBUTES_TAG
{
        /* pixel format type */
        PVRSRV_PIXEL_FORMAT           pixelformat;
        /* dimensions information structure array */
        DISPLAY_DIMS                  sDims;
} DISPLAY_SURF_ATTRIBUTES;
```

DestroyDCSwapChain

```
static PVRSRV_ERROR DestroyDCSwapChain(IMG_HANDLE hDevice, IMG_HANDLE
                 hSwapChain);
```

Inputs

hDevice	Handle to 3rd party private data structure – XXX_DEVINFO
hSwapChain	Handle to swap-chain structure

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_INVALID_PARAMS`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnDestroyDCSwapChain

Description

This function destroys the swap chain specified by the swap chain handle – by undoing the actions performed by CreateDCSwapChain. Associated command queues must also be destroyed.

SetDCDstRect

```
static PVRSRV_ERROR SetDCDstRect(IMG_HANDLE hDevice, IMG_HANDLE hSwapChain,
                    IMG_RECT *psRect);
```

Inputs

hDevice	Handle to 3rd party private data – XXX_DEVINFO
hSwapChain	Handle to swap-chain data structure
psRect	Specifies the region in the destination surface to be 'updated' with data from the source region

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_NOT_SUPPORTED	Not supported

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSetDCDstRect

Description

This function specifies the region in the destination surface to be 'updated' with data from the source region of the swap-chain.

SetDCSrcRect

```
static PVRSRV_ERROR SetDCSrcRect(IMG_HANDLE hDevice, IMG_HANDLE hSwapChain,
                IMG_RECT *psRect);
```

Inputs

hDevice	Handle to 3rd party private data – XXX_DEVINFO
hSwapChain	Handle to swap-chain data structure
psRect	Specifies the region to be 'sourced' from the source region

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_NOT_SUPPORTED	Not supported

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSetDCSrcRect

Description

This function specifies the region in the source surface of the swap-chain used to 'update' with the destination surface.

SetDCDstColourKey

```
static PVRSRV_ERROR SetDCDstColourKey(IMG_HANDLE hDevice, IMG_HANDLE
                    hSwapChain, IMG_UINT32 ui32CKColour);
```

Inputs

hDevice	3rd party private device handle
hSwapChain	Swap Chain handle
ui32CKColour	Colour Key Colour for destination surface. Note: pixel format must match the format of the surface

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_NOT_SUPPORTED`	Function not supported

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSetDCDstColourKey

Description

Specifies colour key for the swap-chain's destination surface.

SetDCSrcColourKey

```
static PVRSRV_ERROR SetDCSrcColourKey(IMG_HANDLE hDevice, IMG_HANDLE
                    hSwapChain, IMG_UINT32 ui32CKColour);
```

Inputs

hDevice	3rd party private device handle
hSwapChain	Swap Chain handle
ui32CKColour	Colour Key Colour for source surface. Note: pixel format must match the format of the surface

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_NOT_SUPPORTED`	Function not supported

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSetDCSrcColourKey

Description

Specifies colour key for the swap-chain's source surface.

GetDCBuffers

```
static PVRSRV_ERROR GetDCBuffers(IMG_HANDLE hDevice, IMG_HANDLE hSwapChain,
                    IMG_UINT32 *pui32BufferCount, IMG_HANDLE *phBuffer);
```

Inputs

hDevice	3rd party private device handle
hSwapChain	Swap Chain handle

Outputs

pui32BufferCount	Number of buffers in swap-chain
phBuffer	Array of handles to buffer data structures (XXX_BUFFER)

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_INVALID_PARAMS	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnGetDCBuffers

Description

Returns an array of XXX_BUFFER handles for the specified swap chain. The primary surface (if present) is represented by the first buffer handle element in the returned array.

Note: the caller already knows the number of buffers in the swap-chain (ui32BufferCount is passed as an argument to CreateDCSwapChain).

SwapToDCBuffer

```
static PVRSRV_ERROR SwapToDCBuffer(IMG_HANDLE hDevice, IMG_HANDLE hBuffer,
               IMG_UINT32 ui32SwapInterval, IMG_HANDLE hPrivateTag,
               IMG_UINT32 ui32ClipRectCount, IMG_RECT *psClipRect);
```

Inputs

hDevice	Handle to 3rd party private data
hBuffer	System physical address of buffer
ui32SwapInterval	Buffer swap to occur after 'swap-interval' vertical blanking periods
hPrivateTag	Private handle
ui32ClipRectCount	Clip Rectangle Count
psClipRect	Pointer Clip Rectangle list

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_INVALID_PARAMS	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSwapToDCBuffer

Description

This function inserts a flip command into the 3rd party driver's command queue. The command will be executed when the command dependencies are met.

SwapToDCSystem

```
static PVRSRV_ERROR SwapToDCSystem(IMG_HANDLE hDevice, IMG_HANDLE
                hSwapChain);
```

Inputs

hDevice	Handle to 3rd party private data
hSwapChain	Handle to swap-chain

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSwapToDCSystem

Description

This function inserts a flip command into the 3rd party driver's command queue to flip the display to the system (primary) surface. The command will be executed when the command dependencies are met.

SetDCState

```
static PVRSRV_ERROR SetDCState (IMG_HANDLE hDevice, IMG_HANDLE ui32State);
```

Inputs

hDevice	Handle to 3rd party private data
ui32State	State control flags word

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the DisplayClass component of the 3rd party driver

Access by services-to-display function pointer

pfnSetDCState

Description

This function sets different states in the 3rd party driver which in turn may be used to perform specific actions, including:

- DC_STATE_FLUSH_COMMANDS – flush the internal vsync queue
- DC_STATE_NO_FLUSH_COMMANDS – clear the flush flag

7. Client Services Display Class API

Client drivers (OGLES, D3D etc.) control the display hardware by calling the Client Services Display Class APIs. This set of functions route through to Kernel Services and, where appropriate, are despatched to the 3rd Party Display driver via a table of function pointers. This section describes the APIs presented to the client drivers.

PVRSRVEnumerateDeviceClass

```
PVRSRV_ERROR PVRSRVEnumerateDeviceClass(PVRSRV_CONNECTION *psConnection,
               PVRSRV_DEVICE_CLASS DeviceClass, IMG_UINT32 *pui32DevCount,
               IMG_UINT32 *pui32DevID );
```

Inputs	
psConnection	Bridge Connection information
DeviceClass	Device Class Type (Display in this case)

Outputs	
pui32DevCount	Number of devices present
pui32DevID	Pointer to an array of Device IDs for each device

Returns	
PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function enumerates 'Device Class' type devices. It is generally called in two phases:

1. pui32DevID==NULL and pui32DevCount==valid pointer. pui32DevCount returns the number of devices in the system
2. Use value returned to allocate an array of IDs and pass address of array in pui32DevID. Valid list of IDs is copied into the array.

The driver can choose from one or more devices based on the ID, 'opening' a given device by passing the correct ID.

PVRSRVOpenDCDevice

```
IMG_HANDLE PVRSRVOpenDCDevice(PVRSRV_DEV_DATA *psDevData, IMG_UINT32
               ui32DeviceID);
```

Inputs

psDevData	Device data information
ui32DeviceID	ID of the device to open

Outputs

Returns

`Valid handle to the device`	Success
`NULL`	Fail

Component

Implemented in the Services Client component

Description

This function ‘opens’ a given display device specified by the device’s ID.

PVRSRVCloseDCDevice

```
PVRSRV_ERROR PVRSRVCloseDCDevice(PVRSRV_CONNECTION    *psConnection,
               IMG_HANDLE hDevice);
```

Inputs

psConnection	Bridge Connection information
hDevice	Handle for the device

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function ‘closes’ a given display device specified by the device’s handle.

PVRSRVEnumDCFormats

```
PVRSRV_ERROR PVRSRVEnumDCFormats (IMG_HANDLE hDevice, IMG_UINT32
               *pui32Count, DISPLAY_FORMAT *psFormat);
```

Inputs

hDevice	Handle for the device

Outputs

pui32Count	Pointer to the number of formats
psFormat	Pointer to a list of formats

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function enumerates the pixel formats for a given display device. Generally, it is called in two phases:

1. pui32Count==valid pointer and psFormat==NULL. The number of formats is returned in pui32Count.
2. psFormat==valid pointer. All pixel formats are copied into the psFormat List

PVRSRVEnumDCDims

```
PVRSRV_ERROR PVRSRVEnumDCDims (IMG_HANDLE hDevice, IMG_UINT32 *pui32Count,
               DISPLAY_FORMAT *psFormat, DISPLAY_DIMS *psDims);
```

Inputs

hDevice	Handle for the device
psFormat	The pixel format to enumerate dimensions for

Outputs

pui32Count	Pointer to the number of dimensions
psDims	Pointer to a list of dimensions

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function enumerates the dimensions (width, height, stride) for a given pixel format and display device. Generally, it is called in two phases:

1. pui32Count==valid pointer and psDims==NULL. The number of dimensions is returned in pui32Count.
2. psDims==valid pointer. All dimension information is copied into the psDims List

PVRSRVGetDCSystemBuffer

```
PVRSRV_ERROR PVRSRVGetDCSystemBuffer(IMG_HANDLE hDevice, IMG_HANDLE
               *phBuffer);
```

Inputs	
hDevice	Handle for the device

Outputs	
phBuffer	Pointer to the system buffer handle

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function retrieves the system buffer handle.

PVRSRVGetDCInfo

```
PVRSRV_ERROR PVRSRVGetDCInfo(IMG_HANDLE hDevice, DISPLAY_INFO*
               psDisplayInfo);
```

Inputs

hDevice	Handle for the device

Outputs

psDisplayInfo	Pointer to the display information

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function retrieves the information about the displays capabilities (see structure below).

```
/* Display info structure definition */
typedef struct DISPLAY_INFO_TAG
{
        IMG_UINT32     ui32MaxSwapChains;
        IMG_UINT32     ui32MaxSwapChainBuffers;
        IMG_UINT32     ui32MinSwapInterval;
        IMG_UINT32     ui32MaxSwapInterval;
        /* physical dimensions of the display required for DPI calc. */
        IMG_UINT32 ui32PhysicalWidthmm;
        IMG_UINT32 ui32PhysicalHeightmm;
        IMG_CHAR       szDisplayName[MAX_DISPLAY_NAME_SIZE];

#if defined(SUPPORT_HW_CURSOR)
        IMG_UINT16     ui32CursorWidth;
        IMG_UINT16     ui32CursorHeight;
#endif

} DISPLAY_INFO;
```

PVRSRVCreateDCSwapChain

```
PVRSRV_ERROR PVRSRVCreateDCSwapChain (IMG_HANDLE hDevice, IMG_UINT32
               ui32Flags, DISPLAY_SURF_ATTRIBUTES     *psDstSurfAttrib,
               DISPLAY_SURF_ATTRIBUTES *psSrcSurfAttrib, IMG_UINT32
               ui32BufferCount, IMG_UINT32 ui32OEMFlags, IMG_UINT32
               *pui32SwapChainID, IMG_HANDLE *phSwapChain);
```

Inputs	
hDevice	Handle for the device
ui32Flags	Create flags (none by default, OEM customisable)
psDstSurfAttrib	Display attributes
psSrcSurfAttrib	backbuffer attributes
ui32BufferCount	Buffer count for the swapchain
ui32OEMFlags	Specified by OEM
ui32BufferCount	Buffer count for the swapchain

Outputs	
pui32SwapChainID	(optional) returns a swapchain ID for cross process swapchain sharing
phSwapChain	Handle to swapchain

Returns	
PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function creates a swapchain on a display device.

PVRSRVDestroyDCSwapChain

```
PVRSRV_ERROR PVRSRVDestroyDCSwapChain (IMG_HANDLE hDevice, IMG_HANDLE
              hSwapChain);
```

Inputs

hDevice	Handle for the device
hSwapChain	Handle to swapchain

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function destroys a swapchain on a display device.

PVRSRVSetDCDstRect

```
PVRSRV_ERROR PVRSRVSetDCDstRect (IMG_HANDLE hDevice, IMG_HANDLE hSwapChain,
              IMG_RECT *psDstRect);
```

Inputs

hDevice	Handle for the device
hSwapChain	Handle to swapchain
psDstRect	Pointer rectangle structure

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function sets the destination rectangle for a given swapchain. This API is only useful in the case of non-fullscreen swapchains.

PVRSRVSetDCSrcRect

```
PVRSRV_ERROR PVRSRVSetDCSrcRect (IMG_HANDLE hDevice, IMG_HANDLE hSwapChain,
              IMG_RECT *psSrcRect);
```

Inputs

hDevice	Handle for the device
hSwapChain	Handle to swapchain
psSrcRect	Pointer rectangle structure

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function sets the source rectangle for a given swapchain, effectively selecting a sub-region of the source to display on the next swap operation.

PVRSRVSetDCDstColourKey

```
PVRSRV_ERROR PVRSRVSetDCDstColourKey (IMG_HANDLE hDevice, IMG_HANDLE
              hSwapChain, IMG_UINT32 ui32CKColour);
```

Inputs	
hDevice	Handle for the device
hSwapChain	Handle to swapchain
ui32CKColour	Colour Key Colour

Outputs

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function sets the destination colour key colour.

PVRSRVSetDCSrcColourKey

```
PVRSRV_ERROR PVRSRVSetDCSrcColourKey (IMG_HANDLE hDevice, IMG_HANDLE
              hSwapChain, IMG_UINT32 ui32CKColour);
```

Inputs

hDevice	Handle for the device
hSwapChain	Handle to swapchain
ui32CKColour	Colour Key Colour

Outputs

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function sets the source colour key colour.

PVRSRVGetDCBuffers

```
PVRSRV_ERROR PVRSRVGetDCBuffers(IMG_HANDLE hDevice, IMG_HANDLE hSwapChain,
               IMG_HANDLE *phBuffer);
```

Inputs

hDevice	Handle for the device
hSwapChain	Handle to swapchain

Outputs

phBuffer	Pointer to an array of handles for swapchain's buffers

Returns

PVRSRV_OK	Success
PVRSRV_ERROR_GENERIC	Fail

Component

Implemented in the Services Client component

Description

This function retrieves the handles for all buffers within a given swapchain

PVRSRVSwapToDCBuffer

```
PVRSRV_ERROR PVRSRVSwapToDCBuffer (IMG_HANDLE hDevice, IMG_HANDLE hBuffer,
               IMG_UINT32 ui32ClipRectCount, IMG_RECT *psClipRect,
               IMG_UINT32 ui32SwapInterval, IMG_HANDLE hPrivateTag);
```

Inputs	
hDevice	Handle for the device
hBuffer	Handle to buffer to swap to
ui32ClipRectCount	Number of clip rectangles to apply
psClipRect	Pointer to a list of clip rectangles
ui32SwapInterval	Number of Vsync intervals between swaps
hPrivateTag	(optional) OEM specific token to be passed through the software stack. Example use-case: audio synching

Outputs

Returns	
`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function ‘queues’ a swap of the display to the specified buffer, passing a list of clip rectangles, swap interval and private tag

PVRSRVSwapToDCSystem

```
PVRSRV_ERROR PVRSRVSwapToDCSystem (IMG_HANDLE hDevice, IMG_HANDLE
               hSwapChain);
```

Inputs

hDevice	Handle for the device
hSwapChain	Handle to swapchain

Outputs

Returns

`PVRSRV_OK`	Success
`PVRSRV_ERROR_GENERIC`	Fail

Component

Implemented in the Services Client component

Description

This function ‘queues’ a swap of the display to the system buffer, passing the swapchain with which the swap is associated.

8. Use Case Example

This section presents a 'use case' example in which a 3rd party display controller and its device driver are integrated into the Consumer Services and controlled via the Display Class APIs. This 'use case' is based on the Services unit test example, Services_test.

8.1. Dynamic Initialisation

Kernel Services has no knowledge of the 3rd party devices in the system until they connect to services. The 3rd party display kernel device driver must unconditionally register with kernel services at initialisation.

After registration, the client driver will be able to call PVRSRVEnumerateDeviceClass and PVRSRVOpenDCDevice

Note: the 'software component' responsible for causing the 3rd party display device to register with kernel services may vary depending on the system and/or environment, e.g. in WinXP the entry point for the 3rd party driver is: DriverEntry (which will call Init).

8.1.1. Initialisation sequence

This example describes the call sequence of a client driver initialising the display using the 3rd party display class API. Other kernel services APIs used by the client driver are not considered here. Initialisation generally starts at driver load time (note: the 3rd party kernel driver is loaded *after* kernel services). The 3rd party driver does the following at initialisation.

- Allocates and sets up the device's private data structure
- Initiates a connection to kernel services by calling OpenPVRServices
- Acquires the kernel services Display Class jump table, enabling calls from the 3rd party device into kernel services.
- Sets up its own Display Class jump table, enabling calls from kernel services into the 3rd party driver.
- Registers the device as a display class device type with kernel services by calling pfnPVRSRVRegisterDCDevice.
- Registers the device's private flip/swap command handler with kernel services command queue manager by calling pfnPVRSRVRegisterCmdProcList.
- Install a Vsync ISR and associated handler function

The client driver makes the following calls:

PVRSRVEnumerateDC – called twice: the first time to enumerate the display devices available, and the second time to get their device IDs.

PVRSRVOpenDCDevice – called once and does the following:

- Finds the matching device node – i.e. matching device class and device ID
- Retrieves a display class information structure (PVRSRV_DISPLAYCLASS_INFO) for the device, from the device node.
- Allocates a syncinfo structure for the device's system surface
- Calls into the 3rd party driver via pfnOpenDCDevice to acquire a handle to the device's private data structure.
- Returns a handle to PVRSRV_DISPLAYCLASS_INFO, which contains the handle to the 3rd party devices private data structure.

Now that the client driver has a handle to the display device it can call other display class functions. Here are some of the other display class APIs that may be called:

PVRSRVGetDCInfo – returns a pointer to the 3rd party display driver's DISPLAY_INFO structure

PVRSRVEnumDCFormats – The first call to this function will return the number of pixel formats and the second call will return an array of pixel format structures.

PVRSRVEnumDCDims – Gets 'dimensions' information

PVRSRVGetDCSystemBuffer – Gets a handle to the primary surface

PVRSRVCreateDCSwapChain – Creates a swap-chain

PVRSRVGetDCBuffers – Get the swap-chain buffer handles

PVRSRVSwapToDCBuffer – Flip the display to the specified buffer

PVRSRVSwapToDCSystem – Flip the display to the primary surface

PVRSRVCloseDCDevice – Close the connection to the 3rd party device

Appendix A. Driver Dynamic Load and Registration

The loading of the 3rd party driver and Services registration can be done dynamically. Unload and the call to pfnPVRSRVRemoveDCDevice can also be done dynamically but the pfnPVRSRVRemoveDCDevice call will fail if client applications still have open connections to the DC driver device.