TEXAS INSTRUMENTS

§ CSSD DESIGN SPECIFICATION

OpenMAX™ 1.0 Nucleus® LOCOSTO

Document Revision: 0.1

Issue Date: 25 January 2006

MakingWireless

TI Proprietary Information — Internal Data

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

OMAP™ is a Trademark of Texas Instruments Incorporated
OMAP-Vox™ is a Trademark of Texas Instruments Incorporated
Innovator™ is a Trademark of Texas Instruments Incorporated
Code Composer Studio™ is a Trademark of Texas Instruments Incorporated
DSP/BIOS™ is a Trademark of Texas Instruments Incorporated
eXpressDSP™ is a Trademark of Texas Instruments Incorporated
TMS320™ is a Trademark of Texas Instruments Incorporated
TMS320C28x™ is a Trademark of Texas Instruments Incorporated
TMS320C6000™ is a Trademark of Texas Instruments Incorporated
TMS320C5000™ is a Trademark of Texas Instruments Incorporated
TMS320C2000™ is a Trademark of Texas Instruments Incorporated
OpenGL® is a Registered Trademark of the Khronos Group
OpenML® is a Registered Trademark of the Khronos Group
OpenVG™ is a Trademark of the Khronos Group
OpenMAX™ is a Trademark of the Khronos Group

All other trademarks are the property of the respective owner.

Copyright © 2005 Texas Instruments Incorporated. All rights reserved.

Information in this document is subject to change without notice. Texas Instruments may have pending patent applications, trademarks, copyrights, or other intellectual property rights covering matter in this document. The furnishing of this document is given for usage with Texas Instruments products only and does not give you any license to the intellectual property that might be contained within this document. Texas Instruments makes no implied or expressed warranties in this document and is not responsible for the products based from this document.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Table of Contents

Table of Contents ..... **i**
List of Figures iii
List of Tables iii
Revision History ..... **iv**
Approvals ..... **iv**
1 Introduction **1**
Purpose 1
Scope1
File Path 1
File Name 1
References 1
Definitions 1
2 Architectural Overview **3**
2.1 System diagram 3
2.2 Architecture diagram 4
2.3 Software Design Interfaces 6
2.4 Features 6
3 Design Rationale **6**
3.1 Relevant Specifications 7
3.2 Design Trade-offs 7
3.3 Hardware Dependencies 7
3.4 Other Pertinent Design Issues 7
4 Memory Requirements **8**
4.1 Memory Allocation 8
5 Sub-Components **9**
5.1 Include files for the parent application 9
5.2 Include files for the OMX SSL Client 9
6 Control and Data flow **10**
6.1 Component States 10
6.2 Component Phases 14
*6.2.1 Component Load (Transition Invalid **à** Loaded)* *14*
*6.2.2 Component Unload (Transition Loaded **à** Invalid)* *14*
*6.2.3 Component Initialization (Transition Loaded **à** Idle)* *15*
*6.2.4 Component Execution (Transition Idle **à** Executing)* *17*
*6.2.5 Component Stop (Transition Executing **à** Idle)* *20*
*6.2.6 Component De-Initialization (Transition Idle **à** Loaded)* *21*
*6.2.7 Component Pause (Transition Executing **à** Pause)* *22*
*6.2.8 Component Resume (Transition Pause **à** Executing)* *23*
*6.2.9 Component Stop (Transition Pause **à** Idle)* *23*
6.3 Input and Output Buffer Allocation Scenarios 24
7 Software Requirements **25**
8 Requirements Traceability **25**
8.1 Defined Types 25
8.1.1 OMX Basic Data Types *25*
8.1.2 SSL Core Data Types *25*
8.2 Data Structures 27
8.2.1 Component private data structure *27*
8.2.2 SSL Plane Query structure *27*
8.2.3 SSL Plane Configuration Structure *28*

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

8.2.4 SSL Plane Handle Structure 28
8.2.5 SSL Display Query Structure 29
8.2.6 SSL Display Configuration Structure 29
8.2.7 SSL Display Deives Listing Structure 30
8.3 API Requirements Coverage 30
8.3.1 Common pre conditions 30
8.3.2 OMX_SSL_ComponentInit 30
8.3.3 OMX_SSL_SetCallbacks 31
8.3.4 OMX_SSL_GetComponentVersion 32
8.3.5 OMX_SSL_SendCommand 32
8.3.6 OMX_SSL_GetParameter 33
8.3.7 OMX_SSL_SetParameter 34
8.3.8 OMX_SSL_GetConfig 35
8.3.9 OMX_SSL_SetConfig 36
8.3.10 OMX_SSL_GetState 37
8.3.11 OMX_SSL_EmptyThisBuffer 38
8.3.12 OMX_SSL_FillThisBuffer 38
8.3.13 OMX_SSL_ComponentTunnelRequest 39
8.3.14 OMX_SSL_ComponentDeInit 39
8.4 Application callbacks 40
8.4.1 EventHandler 40
8.4.2 EmptyBufferDone 41
8.4.3 FillBufferDone 41
8.5 Internal Functions 42
8.5.1 SSL Core Task 42
8.6 Non-API Requirements Coverage 42
9 Assumptions 43
10 Appendix A – Direct Screen Access (DSA) 43

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

List of Figures

Figure 1 System diagram 3
Figure 2 OMX SSL Client architecture diagram 4
Figure 3 Multiple Plane Creation 5
Figure 4 Component State Diagram 10
Figure 5 OMX Component state transitions and SSL Core Task Transitions 12
Figure 6 OMX Component state transitions and SSL Core Task transitions 13
Figure 7 OMX component load 14
Figure 8 OMX component unload 15
Figure 9 OMX component initialization 16
Figure 10 OMX component Initialization: Plane Creation in SSL Core 17
Figure 11 OMX component execution: Idle to Executing transition 19
Figure 12 OMX component stop 20
Figure 13 OMX component de-initialization: Plane Destroy process 21
Figure 14 OMX component execution: SSL Pausing 23
Figure 15 OMX Component Pause: Return to Execute state 23
Figure 16 OMX Component Pause: Return to Idle state 24

List of Tables

Table 1 Terms and Acronyms 2
Table 2 Memory requirements of the **OMX SSL Client** 8
Table 3 Include files for BMI/MMI 9
Table 4 Include files for the **OMX SSL Client** 9
Table 5 State transitions in the **OMX SSL Client** 10
Table 6 Display Formats Supported and Bit position for query 26
Table 7 OMX_SSL_COMPONENT_PRIVATE_DATA structure 27
Table 8 OMX_SSL_PLANE_QUERY instance structure 27
Table 9 OMX_SSL_PLANE_CONFIG instance structure 28
Table 10 OMX_SSL_PLANE_HANDLE_TYPE structure 28
Table 11 OMX_SSL_DISPLAY_QUERY_TYPE structure 29
Table 12 OMX_SSL_DISPLAY_CONFIG_TYPE structure 29
Table 13 OMX_SSL_AVAILABLEDISP_TYPE structure 30

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Revision History

REV	DATE	AUTHOR	NOTES
0.1	25th January 2006	J Raghuram Karthik	First Version

Approvals

REV	APPROVAL 1	DATE	APPROVAL 2	DATE
0.1	Rajan Narendran		Prabhavathy S	

Please read the “Important Notice” on the next page.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

1 Products		2 Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2005, Texas Instruments Incorporated

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

1 Introduction

This document describes the design of the OpenMAX™ 1.0 compliant Screen Services Layer (SSL) Component for the Locosto Imaging Subsystem. The typical operating environment for the SSL Component would be:

- Nucleus® Operating System on LOCOSTO™'s ARM7 processor.
- OpenMAX™ 1.0 core to expose a standardized API to the application.
- I-SAMPLE BOARD.
- Generic Protocol Framework (GPF) for System resource access.

Purpose

This document details the design specifications for OpenMAX™ 1.0 **OMX SSL Client** on LOCOSTO.

Scope

This document addresses only design specifications.

Additional technical data can be found by referring to the OMAP™SS&P Technical Perspective and Data Package document.

The document provides information about technical data artifacts, including their title, standard ClearCase® VOB location, a brief description and the System or Software Checkpoint where the artifact is first introduced into the development process.

File Path

This design specification document shall be captured in ClearCase® path defined in the project CM Plan:

```
\OMAPSW_SysDev\LOCOSTO\Multimedia\System_Core\Docs
```

File Name

The file name of this document is OMAPSSP_LOCOSTO_OMX_SSL_DesignSpec.doc.

References

All References can be found on the Cellular Systems web site or the World Wide Process and Tools Group web site.

Definitions

Terms used in this document can be found in the Cellular Systems Glossary Document.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Terms that are introduced in this document are detailed below:

Table 1 Terms and Acronyms

ACRONYM	DEFINITION
DSP	Digital Signal Processor
GPP	General Purpose Processor
OMX	OMX and OpenMAX™ 1.0 are used interchangeably in the document.
API	Application Programming Interface
ARM	Advanced RISC Machines
OSAL	Operating System Adaptation Layer
UI	User Interface
DMM	Dynamic Mapped Memory

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

2 Architectural Overview

The Screen Services Layer (SSL) is middle-level layer that provides for an OpenMAX™ 1.0 compliant interface to applications for the display feature on the LOCOSTO™ platform. The operating environment for the SSL consists of the Nucleus® operating system and the Generic Protocol Framework (GPF). The application layer uses the SSL for all of the display functionalities. With respect to the Imaging Subsystem, the SSL operates in tandem with the IMG_Client and the CAMD_Client Components to provide for the view-finder and capture features. The SSL Layer makes use of the IMG_Client component to compose the multiple planes that it gets to the final frame buffer. At the lowest level, the SSL interacts with the LCD Manager to access the physical display.

2.1 System diagram

Figure 1 shows the architecture of **OMX SSL Client** in context of the imaging sub-system.

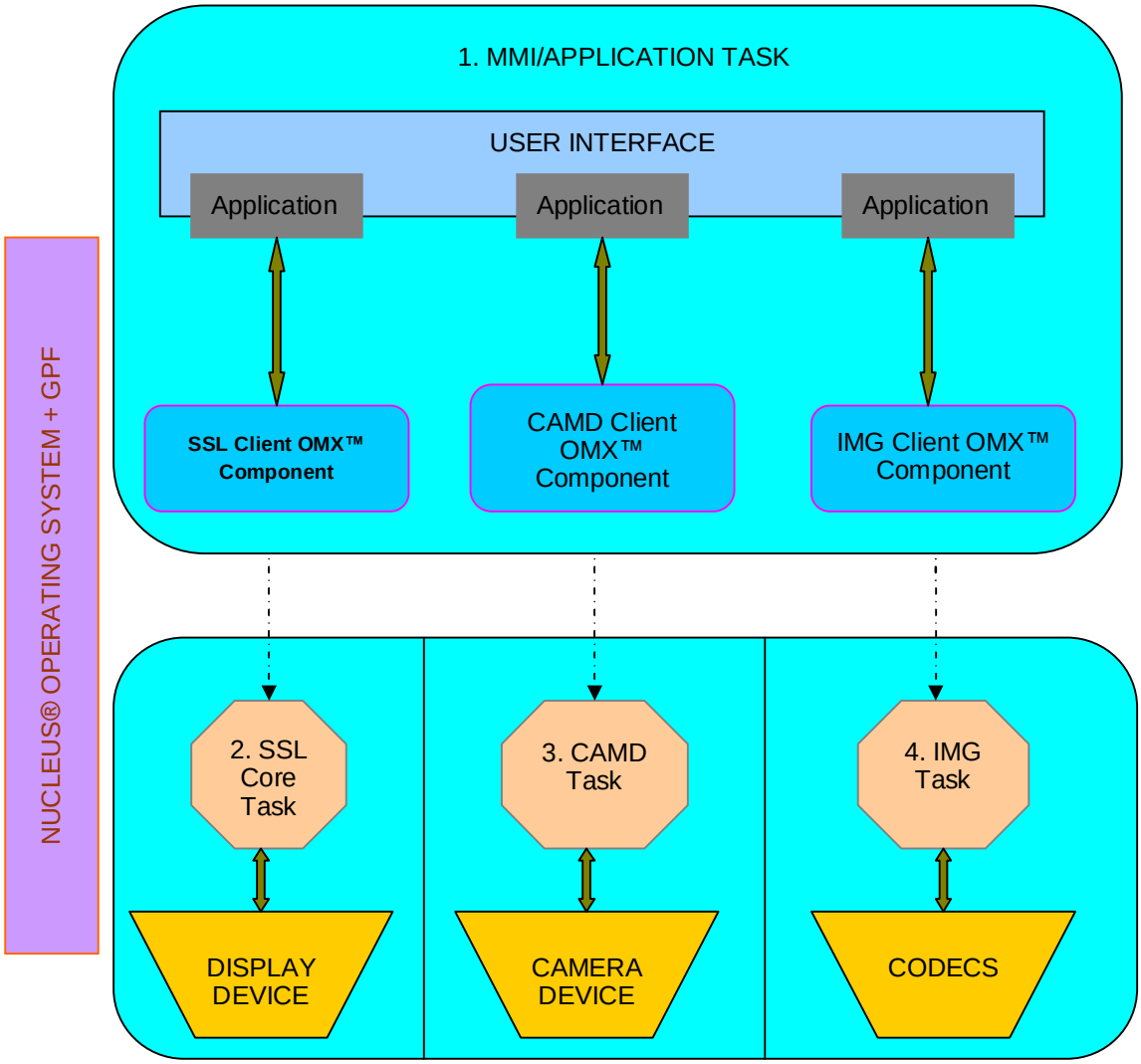


Figure 1 System diagram

From the diagram, it is seen that there are 4 tasks in the system, namely:

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

1. MMI/Application Task
2. SSL Core Task (includes the LCD Manager)
3. Camera Driver Task (CAMD)
4. Imaging Wrapper (IMG) Task

The SSL Task is actually the LCD Manager Task with added functionality. The CAMD Task is the Camera Driver task with enhanced functionality. The IMG Task includes the in it.

All these tasks make use of the GPF for using the system resources. The GPF abstracts the functionalities offered by the operating system.

2.2 Architecture diagram

Figure 2 shows the architecture of the OMX SSL Client, as well as its interfaces with the MMI/Application Layer. The OMX SSL Client runs in the same task context as the MMI/Appliacation Layer. The SSL Core executes as a separates task and includes the LCD Manager in it.

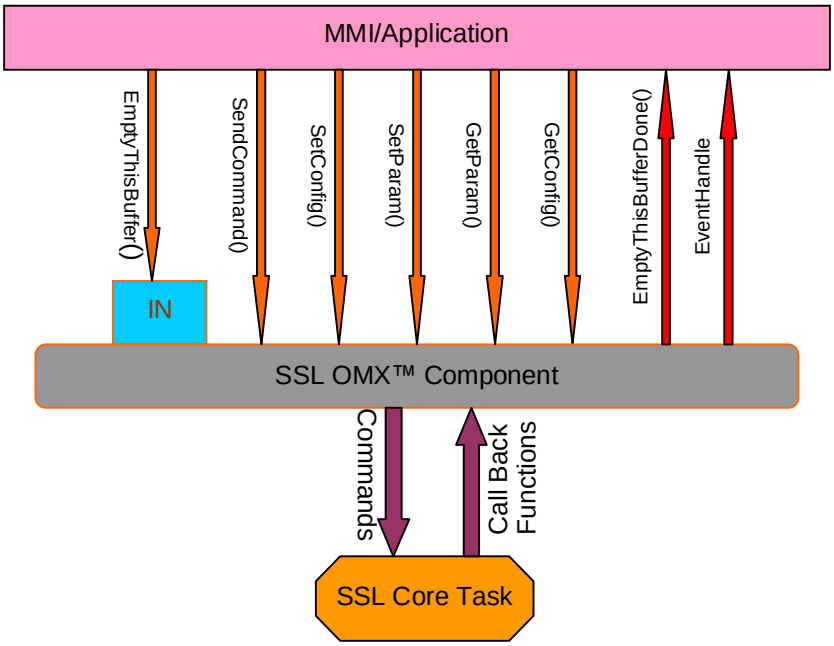


Figure 2 OMX SSL Client architecture diagram

Figure 3 indicates multiple applications each creating an instance of the OMX SSL Client for displaying a ‘plane’. A ‘plane’ as will be described later, is a logical region of disaply access to the physical display device.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

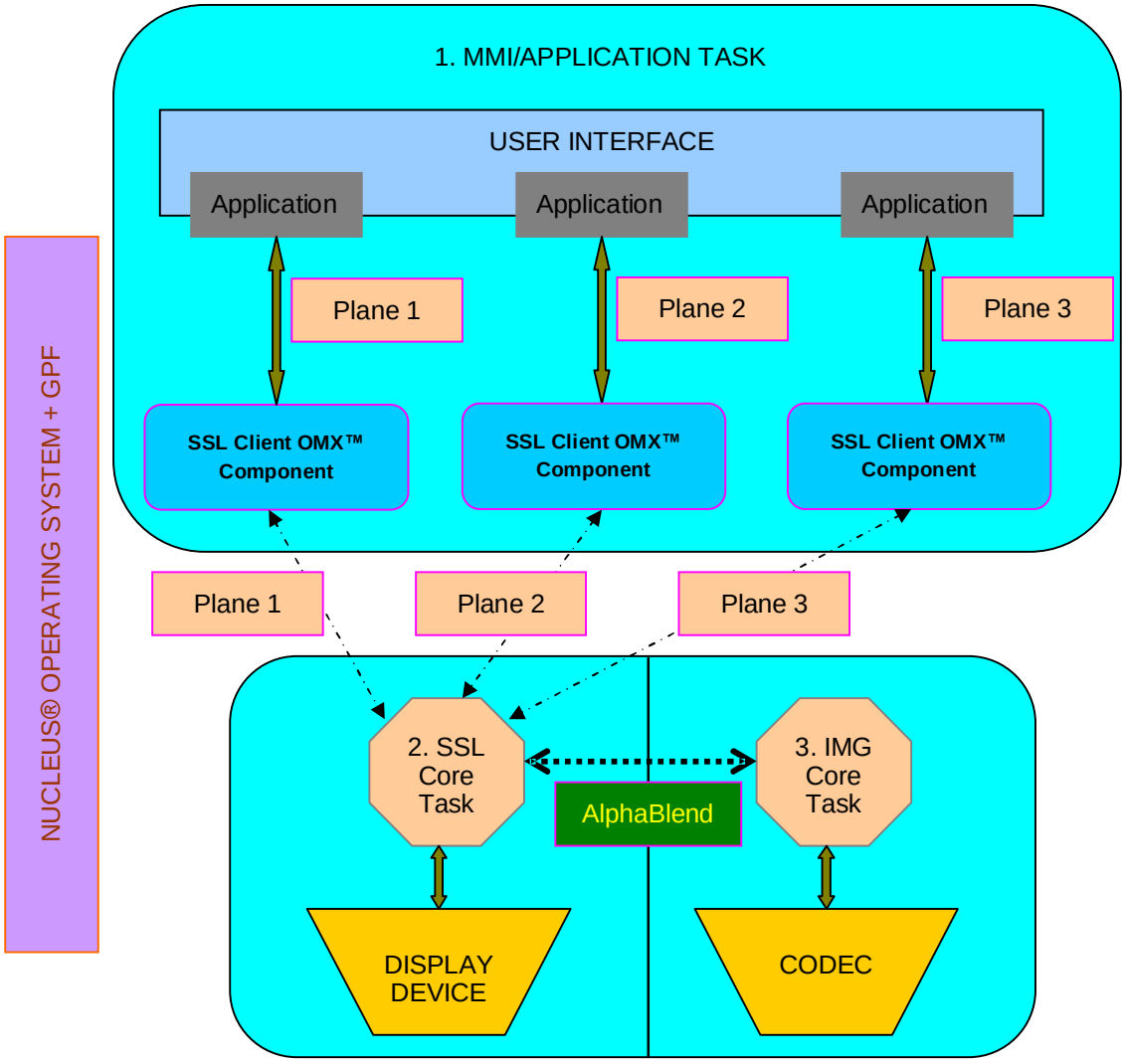


Figure 3 Multiple Plane Creation

Each application that requests a plane creates a new SSL OMX component configures it and uses it.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

2.3 Software Design Interfaces

The SSL OMX™ component exposes the standard OMX Core APIs to the application. These are disussed in Section 8.3. The component maps these functions to SSL Core Task functionalities through messages and obtains responses through callbacks. It provides response to the application layer through call backs wherever applicable as well.

2.4 Features

The SSL is a generic display manager layer. It uses the services of the IMG and LCD Manager to provide for multiple logical planes and combines them to a single display. The interface exposed by SSL to applications is Open MAX 1.0 compliant. Following are the key features of the SSL:

- **Multiple Plane Handling**
- **Screen Composition using Alpha Blending**
- **Always on Top plane specification**
- **Background configuration**
- **Direct Screen Access (DSA)**

3 Design Rationale

The OMX SSL Component is a **display sink**. This means that, IN OMX terminology, it has only input ports. The SSL Compoent provides a single input port on which the application can provide the image data to be displayed.

An SSL Component instance is associated with 1) A Physical Display Device that it shares with other applications 2) A logical display plane that is private to this SSL instance.

A plane is a memory region mapped to a screen area. It is characterized by the following properties:

1. **Application Data Pointer**: The ‘input’ data pointer provided by the application
2. **SSL Shadow Buffer pointer**: The copy of the Application Data that the SSL maintains. The SSL Core task blends the shadow buffers into the actual frame buffer and displays it.
3. **Always on Top Flag**: This is a flag that indicates if this plane is an ‘Always on Top’ frame. This feature is useful for On Screen Display functionality. Always on Top planes are the last to be blended to form the frame buffer. The blending performed is binary blending with the existing frame buffer. In case, more than one plane is ‘Always on Top’, then the most recently activated plane takes precedence.
4. **Alpha Parameter**: This is the alpha blending parameter for this plane. This is used to blend this plane with the current frame buffer.
5. **Offset** on the Screen comprising of: X offset and Y Offset. The Offset is measured with the top left being considered (0, 0).
6. **Dimensions** comprising of: X Length and Y Length (Width and Height)
7. **Active Flag**: This is a flag indicates if this plane is ‘Active’ or ‘Inactive’. All ‘Active’ planes are considered for blending into the framebuffer in the display scans. A plane may be deactivated for updating the contents.

A display scan is where the ‘Active’ queue is examined in the SSL Core and blended to form the FrameBuffer. After the ‘Active Queue’ scan is completer, the ‘Active Always on Top’ Queue is examined and the planes are binary blended with the current framebuffer.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

An exception to the plane described arises in the case of Direct Screen Access (DSA). When a DSA is requested by the application, the SSL Core returns the FrameBuffer pointer to the application through the OMX Component. The application uses this to dump the display data. Thus a DSA Plane has the framebuffer pointer, the offset and dimensions and the active flag associated with it. Even if a DSA plane is active, other planes can be active and can be blended with the FrameBuffer to produce composite images. The DSA feature is especially useful for ViewFinder type of applications. There can be more than on DSA plane, but only to mutually exculsive regions. This can provide for exciting options like display screen windowing etc.

The SSL OMX Component is associated with only one plane at a time. The SSL Core task is responsible for handling multiple OMX SSL Clients and the associated planes. For more information on the SSL Core, please refer to the CSSD Design Specification for SSL Core Task.

3.1 Relevant Specifications

Refer to the following specifications for additional information:

- OpenMAX™ 1.0 Standard (www.khronos.org)
- CSSD Design Specification for CAMD Client OMX Component
- CSSD Design Specification for IMG Client Component
- CSSD Design Specification for SSL Core Task
- GPF Architecture Documentation

3.2 Design Trade-offs

- The **OMX SSL Client** and MMI/Application run in the same Nucleus® task. This approach has been adopted because a Nucleus® task is a costly resource.
- The **OMX SSL Client** and MMI/Application run in the same Nucleus® task and the **OMX SSL Client** is not permitted to block the task. This means the **OMX SSL Client** is passive and can perform its operations only when it is given control by BMI/MMI.
- The use of DSA provides for a fast mechanism for the application to display frames, but restricts the number of planes that can be simultaneously displayed.

3.3 Hardware Dependencies

This component is designed to run on the Locosto platform. The development board used was an I-Sample Revision 3 board.

3.4 Other Pertinent Design Issues

There are no design issues.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

4 Memory Requirements

Memory requirements of the **OMX SSL Client** are listed in Table 2.

Table 2 Memory requirements of the **OMX SSL Client**

Type	Size	Structure	Comments

From the above table, the total dynamic memory required for the **OMX SSL Client** can be obtained using the following calculation. Sizes of buffer headers and buffers for a port will need to be included only if the **OMX SSL Client** allocates buffers for that port.

Total dynamic memory size in bytes =

4.1 Memory Allocation

The MMI/Application shall allocate the memory region for the data to be displayed except in the case of using DSA. The Application shall request a plane to be created for the data it wishes to display. The plane properties are provided under section 3. It is important to note that in the DSA mode, planes need to occupy mutually exclusive regions of memory.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

5 Sub-Components

5.1 Include files for the parent application

The BMI/MMI, which uses the **OMX SSL Client**, must include the files listed in Table 3.

Table 3 Include files for BMI/MMI

File	Function
OMX_Core.h	Contains prototypes of the core functions and definitions used by both the application and the component to access common items.
OMX_Component.h	Contains the definitions used to define the public interface of a component. This header file is to be used by both the application and the component.
OMX_Index.h	Contains the definitions of parameter and configuration indices for both applications and components.
OMX_Image.h	Contains structures needed by Image applications to exchange parameters and configuration data with the components.
OMX_IVCommon.h	Contains structures needed by Video and Image applications to exchange parameters and configuration data with the components.

5.2 Include files for the OMX SSL Client

Table 4 Include files for the **OMX SSL Client**

File	Function
OMX_Core.h	Contains prototypes of the core functions and definitions used by both the application and the component to access common items.
OMX_Component.h	Contains the definitions used to define the public interface of a component. This header file is to be used by both the application and the component.
OMX_Index.h	Contains the definitions of parameter and configuration indices for both applications and components.
OMX_Image.h	Contains structures needed by Image applications to exchange parameters and configuration data with the components.
OMX_IVCommon.h	Contains structures needed by Video and Image applications to exchange parameters and configuration data with the components.
OMX_SSL_component.h	Contains prototypes of internal functions, structures for the private data area and others.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

6 Control and Data flow

6.1 Component States

The **OMX SSL Client** will exist in one of five states at any given time. Component states are controlled by MMI/Application via the OMX_SendCommand macro. Figure 4 represents the state diagram for the **OMX SSL Client**. The OMX core does not maintain the state for the components. The core is involved in two only state transitions; which are from INVALID state to LOADED state (using function OMX_GetHandle) and unloading (using OMX_FreeHandle). The remainders of all state transitions are controlled by MMI/Application via the OMX_SendCommand macro.

Note that state transition from INVALID state to LOADED state refers to the scenario when component is not yet loaded into the memory and BMI/MMI tries to load it using OMX_GetHandle. If component goes in the INVALID state due to corruption of its data structures etc then component cannot change its state. In this case the component should be de-initialized and unloaded and then should be loaded again.

The states are described in OpenMAX™ 1.0 Core Design Specifications. Table 5 describes transitions between various states.

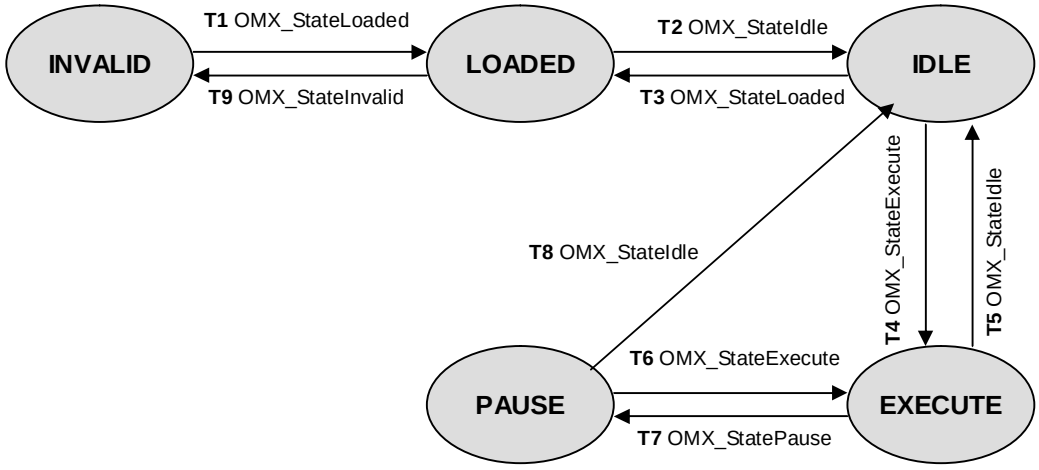


Figure 4 Component State Diagram

Table 5 State transitions in the **OMX SSL Client**

Transition	State Change	Operation
T1	Invalid->Loaded	§ Allocate component's private data area. § Error handling: When the above operation fails, the components state remains as Invalid. § If there were no errors, change component's state to Loaded.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Transition	State Change	Operation
T2	Loaded->Idle	§ Allocate the Shadow Buffer for the SSL Component instance's plane. § Before this is done, the Application needs to configure the plane properties using the SetConfig function. § Error handling: When any of the above operations fails, the components state remains as Loaded. § If there were no errors, change component's state to Idle.
T3	Idle->Loaded	§ Release the Shadow buffer allocated for this instance's plane. § Change component's state to Loaded.
T4	Idle->Execute	§ Change component's state to Execute. § Set this plane's state to 'Active' using the SSL Core Task. § Call the EmptyThisBufferDone after display.
T5	Execute->Idle	§ Change the state of the current plane to 'Inactive' using the SSL Core task. § Change the component state to Idle.
T6	Pause->Execute	§ Change component's state to Execute. § Instruct the SSL Core Task to activate this plane again.
T7	Executing->Pause	§ Change component's state to Pause. § Instruct the SSL Core Task to deactivate this plane.
T8	Pause->Idle	§ Same as transition T5.
T9	Loaded->Invalid	§ Release component's private data area.

Figure 5 shows state transitions in the **OMX SSL Client** and the corresponding **SSL Core** state transitions. Figure 6 shows state transitions in the **OMX SSL Client** and the corresponding **SSL Core** state transitions, when the **OMX SSL Client** enters or leaves its Pause state. Section 6.2 has detailed diagrams indicating various phases during the life cycle of the **OMX SSL Client**.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

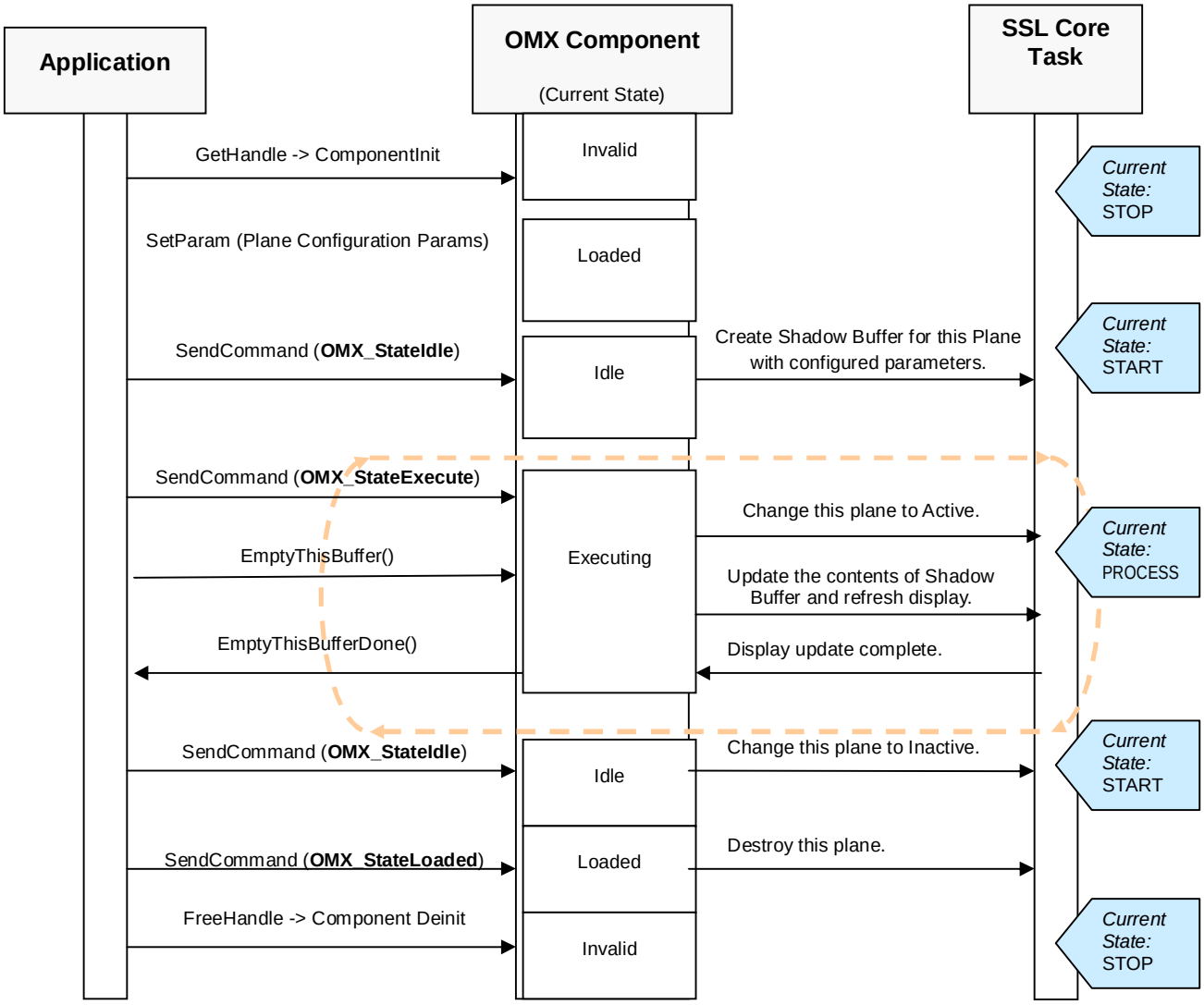


Figure 5 OMX Component state transitions and SSL Core Task Transitions

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

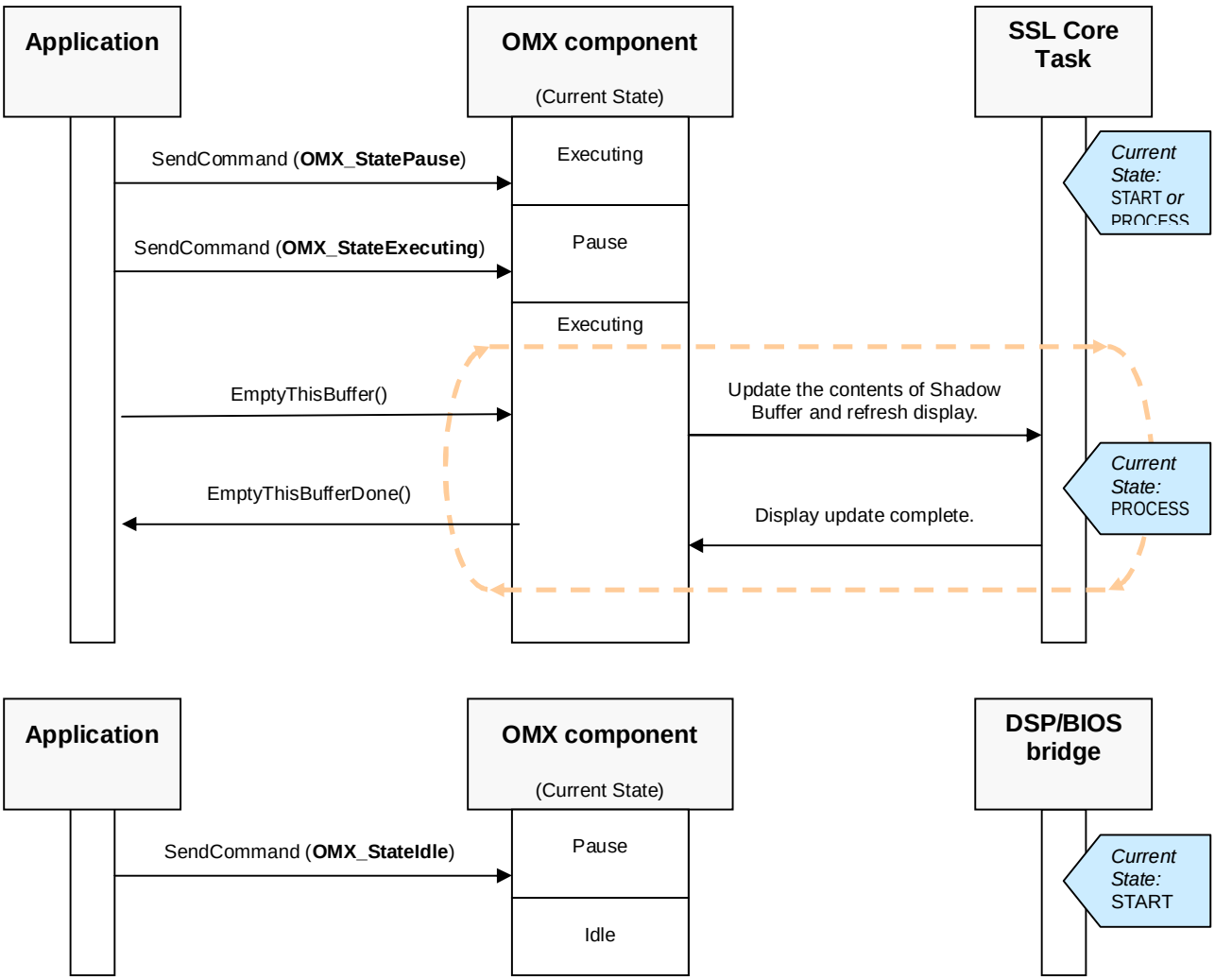


Figure 6 OMX Component state transitions and SSL Core Task transitions

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

6.2 Component Phases

There are various phases in life cycle of the **OMX SSL Client**. Using sequence diagrams, this section describes the control and data flow between BMI/MMI and the **OMX SSL Client**. The OMX Component depicted in these sequence diagrams is the **OMX SSL Client**.

6.2.1 Component Load (Transition Invalid à Loaded)

The **OMX SSL Client** is loaded as described in the OpenMAX™ 1.0 core design specification. During the loading phase, the component performs the following actions:

- Populates the component handle structure with pointers to the functions implemented by it.
- Allocates and initializes its private data area. Contents of the private data area are described in detail in section 8.2.1.

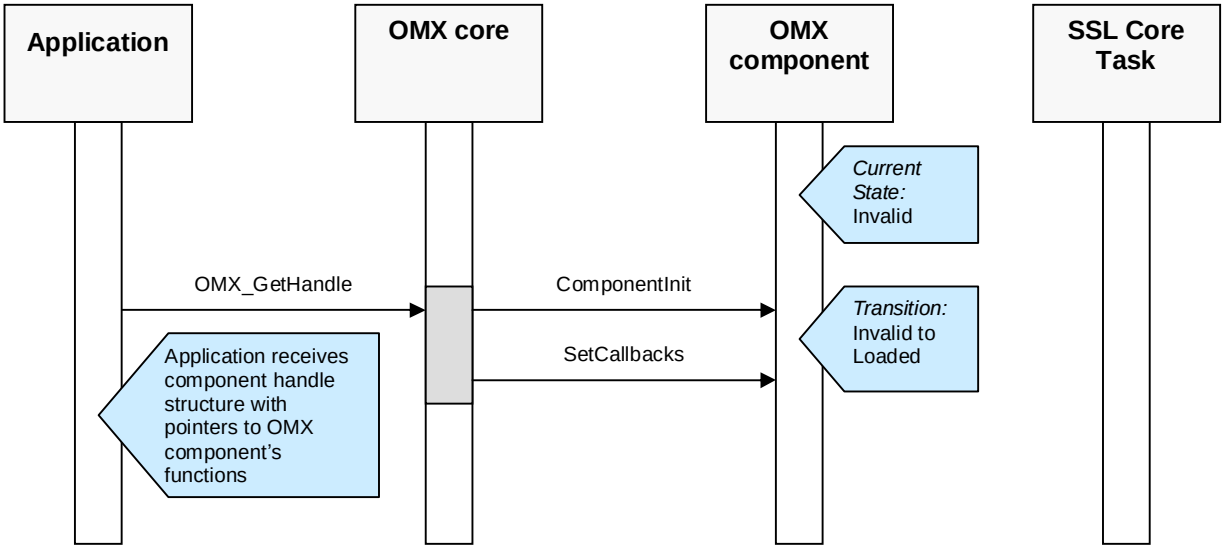


Figure 7 OMX component load

6.2.2 Component Unload (Transition Loaded à Invalid)

The SSL OMX Component is unloaded as described in the OpenMAX™ 1.0 Core design specification. During the unload phase, the component performs the following actions:

- Releases the private data area that it had allocated during the load phase.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

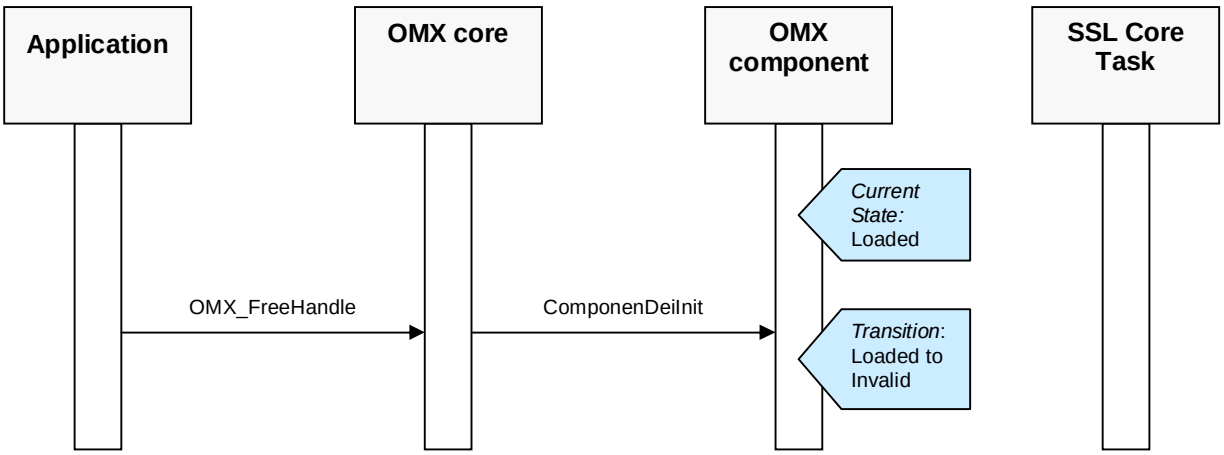


Figure 8 OMX component unload

6.2.3 Component Initialization (Transition Loaded à Idle)

Figure 9 and Figure 10 show the initialization phase of the component. Initialization of the SSL OMX Component proceeds as follows:

- The MMI/Application begins by defining the number of ports and how many buffers it needs on each port. In this case, there shall be only one port – the input port and one buffer.
- The OMX component stores these definitions in its private data area.
- The Application obtains the list of available displays through the OMX_GetParam(OMX_IndexParamSSLDisplayDevicesAvailable) command.
- The Application then sets the active display using the OMX_SetParam(OMX_IndexParamSSLActiveDisplay) command.
- The BMI/MMI sends various parameters, which will be used by the component to initialize the Plane. These are as per the OMX_SSLShadowBufferConfig structure.
- The OMX component stores each parameter into its private data area.
- MMI/Application issues SendCommand with OMX_StateIdle to change the state of the component to Idle
- The component allocates input and output buffers as required. For the SSL, the input buffer shall always be allocated by the application itself. It shall however provide details of the plane it requests so that a shadow buffer may be allocated (Except if DSA is to be used).
- After this, it sends a command to the SSL Core to create a plane (the shadow buffer). The SSL core creates a plane with the given parameters. The default state of this plane is 'inactive'. The OMX Component now has a plane handle for future communications.
- The OMX Component transitions to the Idle state after the successful creation of the Shadow Buffer.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

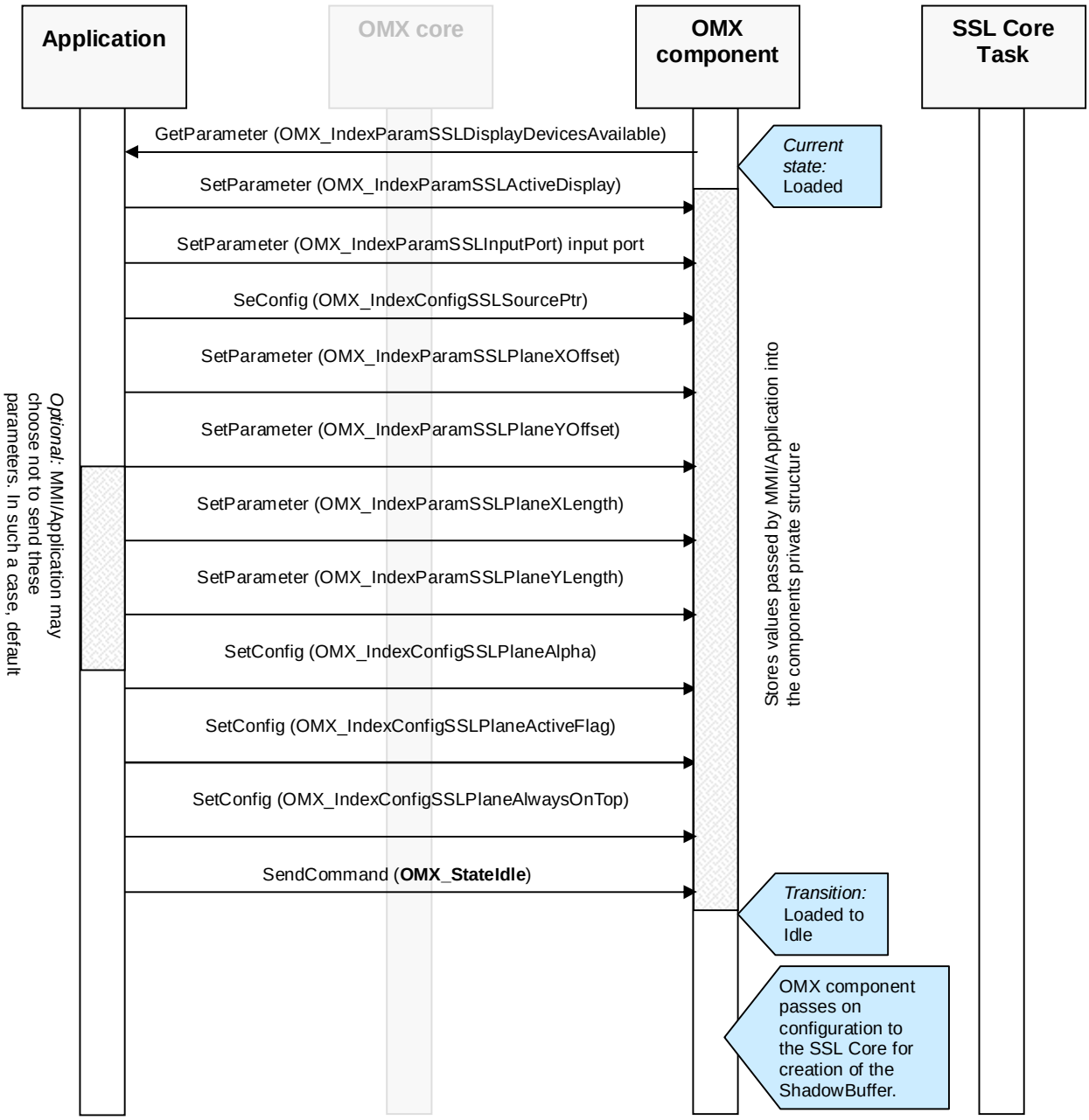


Figure 9 OMX component initialization

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

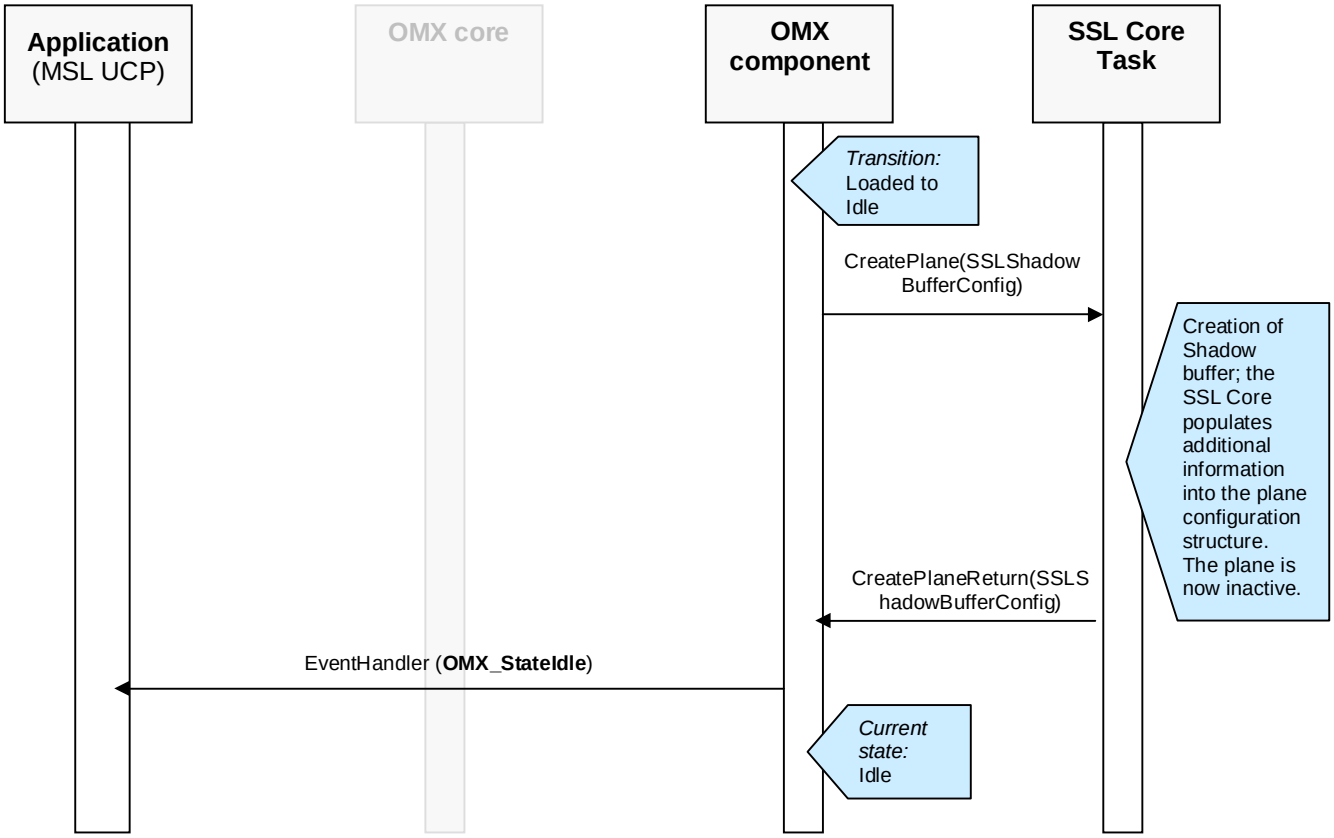


Figure 10 OMX component Initialization: Plane Creation in SSL Core

6.2.4 Component Execution (Transition Idle à Executing)

Prior to component execution, the Application should have initialized the component without errors. After initialization has been completed, the component is ready for execution with all required buffers allocated and linked with appropriate data structures. Component execution proceeds as follows:

- Application issues SendCommand with OMX_StateExecuting to change the state of the component to executing.
- The state of the current plane is set to active. This in turn causes the SSL core to queue the current plane handle in its 'Active' Queue or the 'Always on Top' queue.
- The source pointer for the buffer may be initialized at create phase or at run time. The EmptyThisBuffer call also updates the source pointer for the plane.
- The application then writes input data into the input and invokes EmptyThisBuffer.
- The SSL Core will update the shadow buffer for this plane, and refresh the framebuffer with all the currently active planes.
- For each input buffer, the component calls EmptyBufferDone to indicate that the provided input has been transferred to the shadow buffer and a display update has been performed.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.
- In the execution state, the component shall continue to update the SSL Core with any new data that it may get using an empty this buffer.
- If a configuration parameter change has been performed (using an EmptyBufferDone or a SetConfig), the ShadowBufferParameterStructure is updated and a command is sent to the SSL Core to reflect these changes. The SSL Core updates the structure and calls the refresh cycle and updates the SSL

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

OMX component that this configuration change is complete. The configurable parameters are the Source Pointer, the Alpha for this plane, the Always on Top Flag and the Active Flag. If an active Always on Top plane is no longer configured to be Always on Top, the plane then becomes the plane in the active queue for blending. If an active plane is configured to be Always on Top, the plane becomes the first plane in the Always on Top Queue.

- The OMX Component then can call the EventHandler or the EmptyThisBufferDone functions.

Figure 11 shows the transition from Idle to Executing state and the ensuing events.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

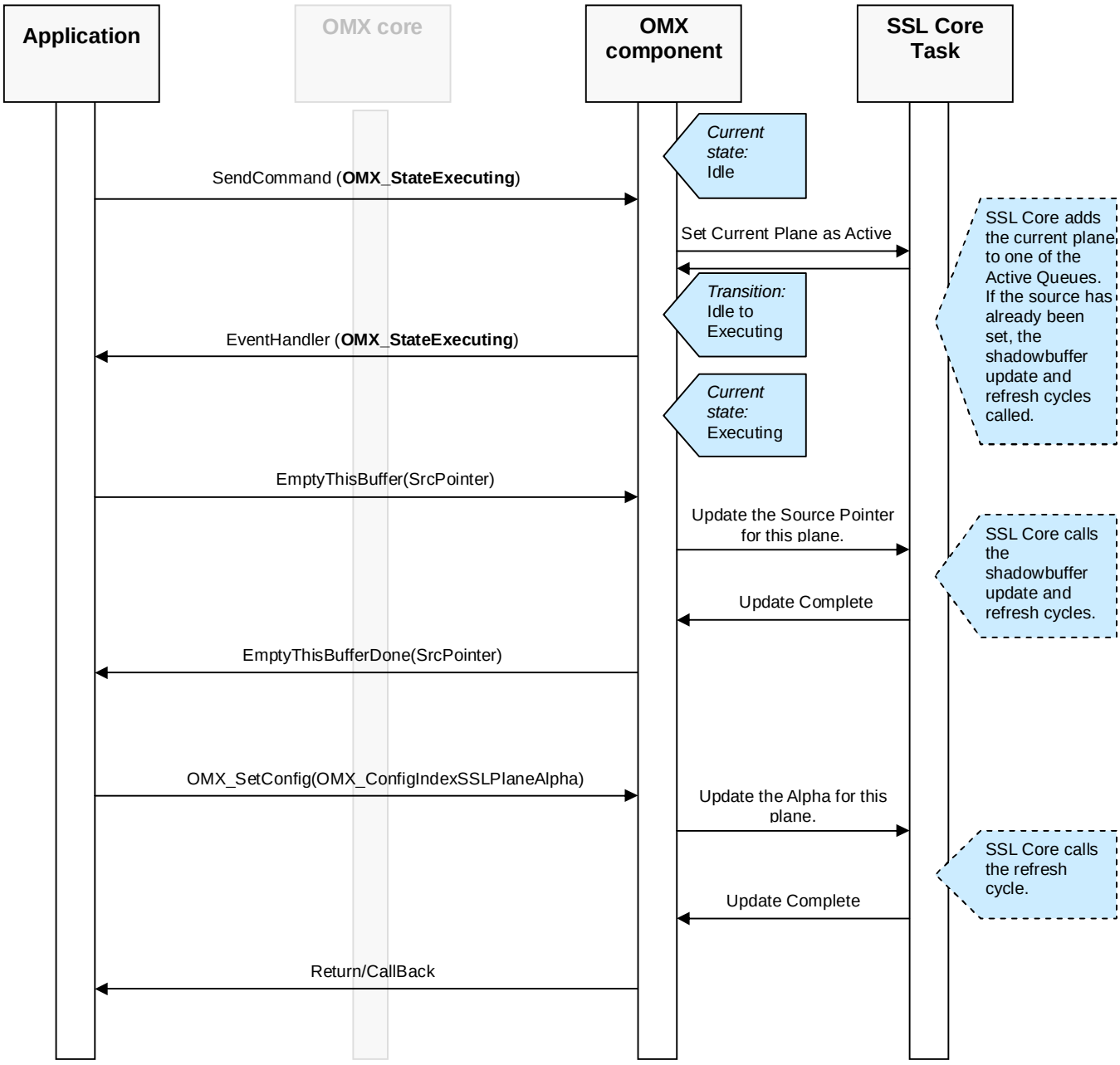


Figure 11 OMX component execution: Idle to Executing transition

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

6.2.5 Component Stop (Transition Executing à Idle)

Component stop proceeds as follows:

- MMI/Application issues SendCommand with OMX_StateIdle
- The SSL Component waits until the SSL has acknowledged the last buffer update request. The SSL Component then sets the current state as inactive and sends a command to the SSL Core informing the core of this change. The SSL core removes the plane from the appropriate queue and informs the SSL Component that the update is complete and performs a refresh cycle.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.

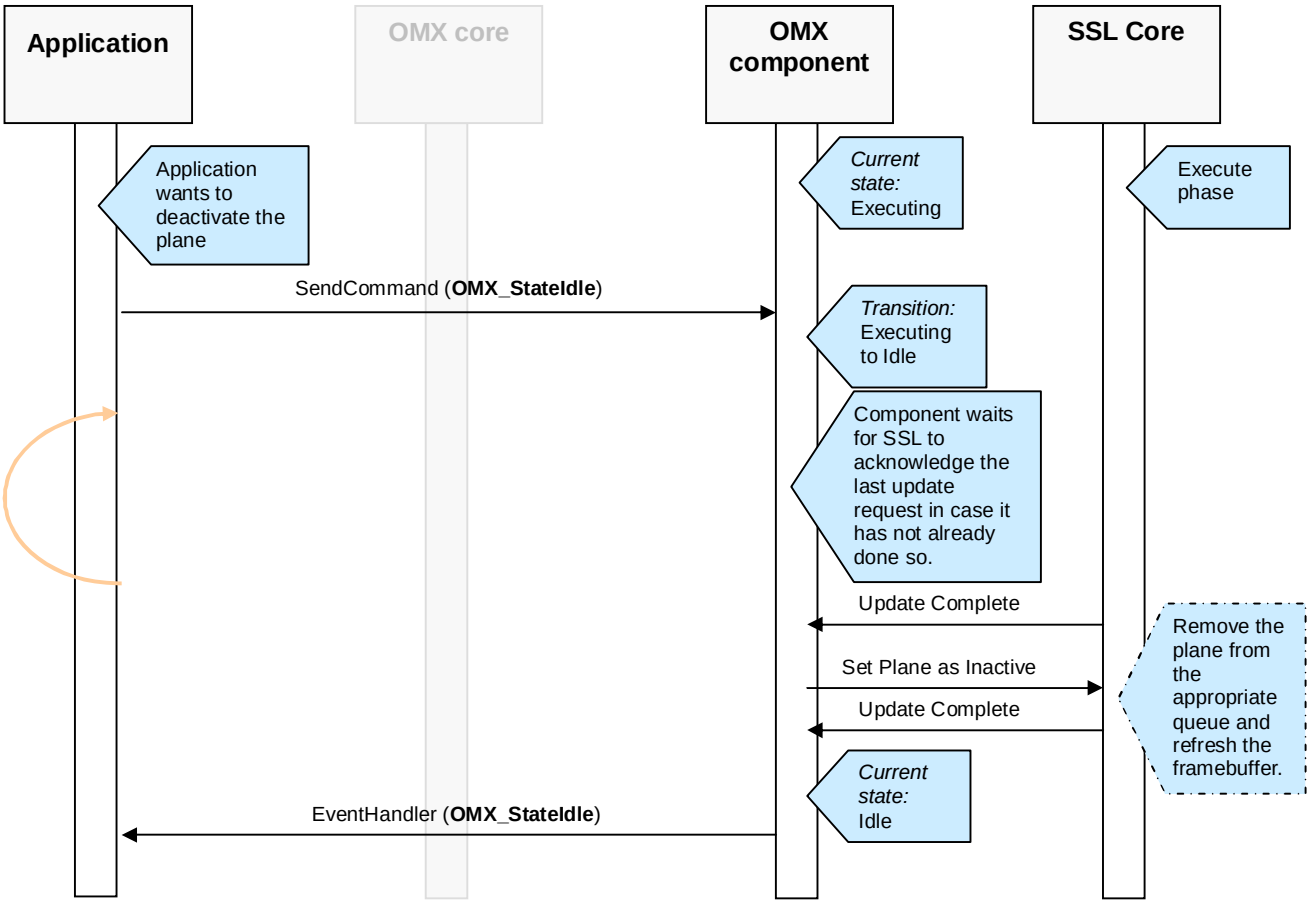


Figure 12 OMX component stop

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

6.2.6 Component De-Initialization (Transition Idle à Loaded)

Component stop proceeds as follows:

- Application issues SendCommand with OMX_StateLoaded to change the state of the component to Loaded
- The component instructs the SSL Core to destroy the plane. The SSL Core acknowledges the destruction.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.

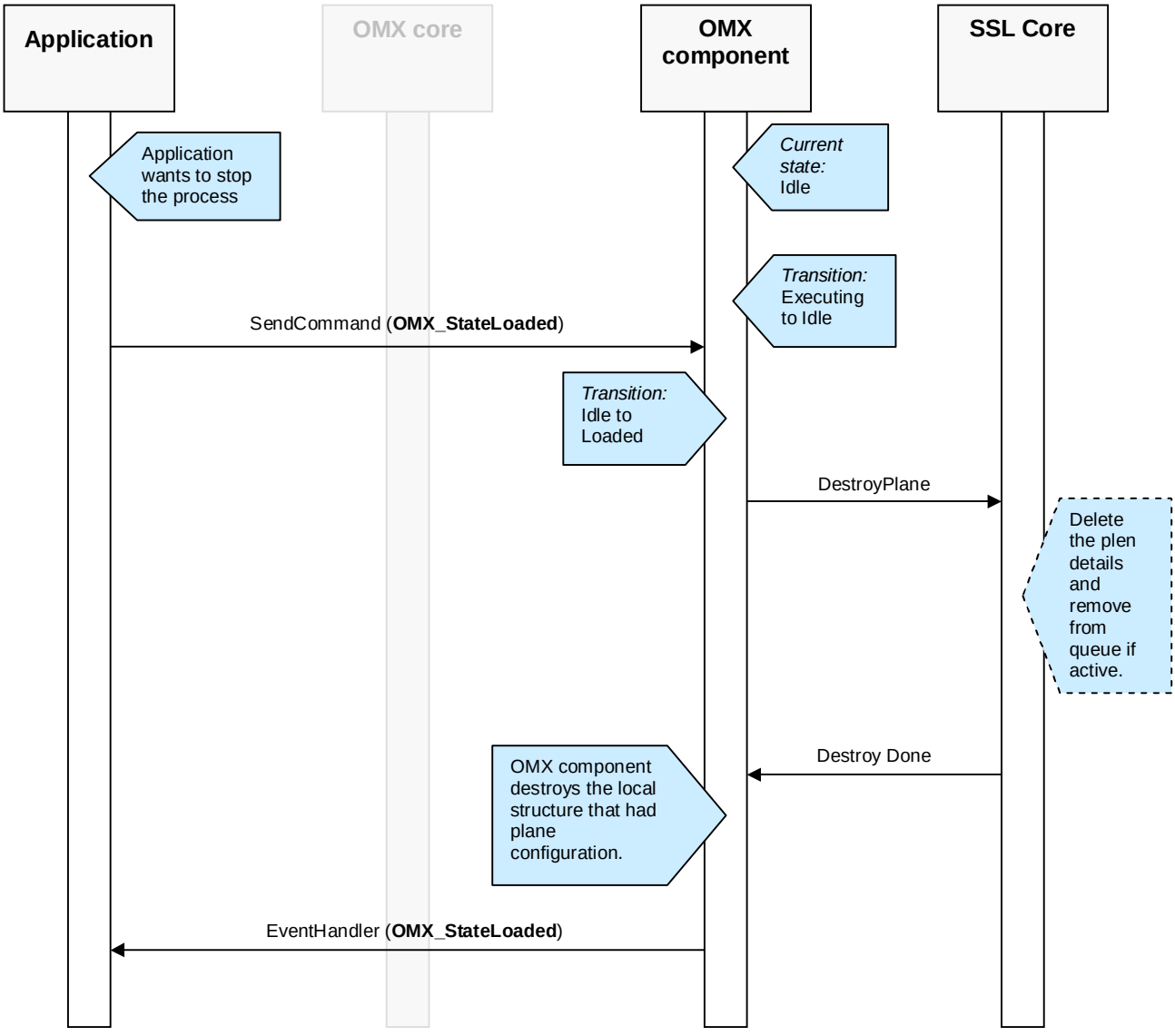


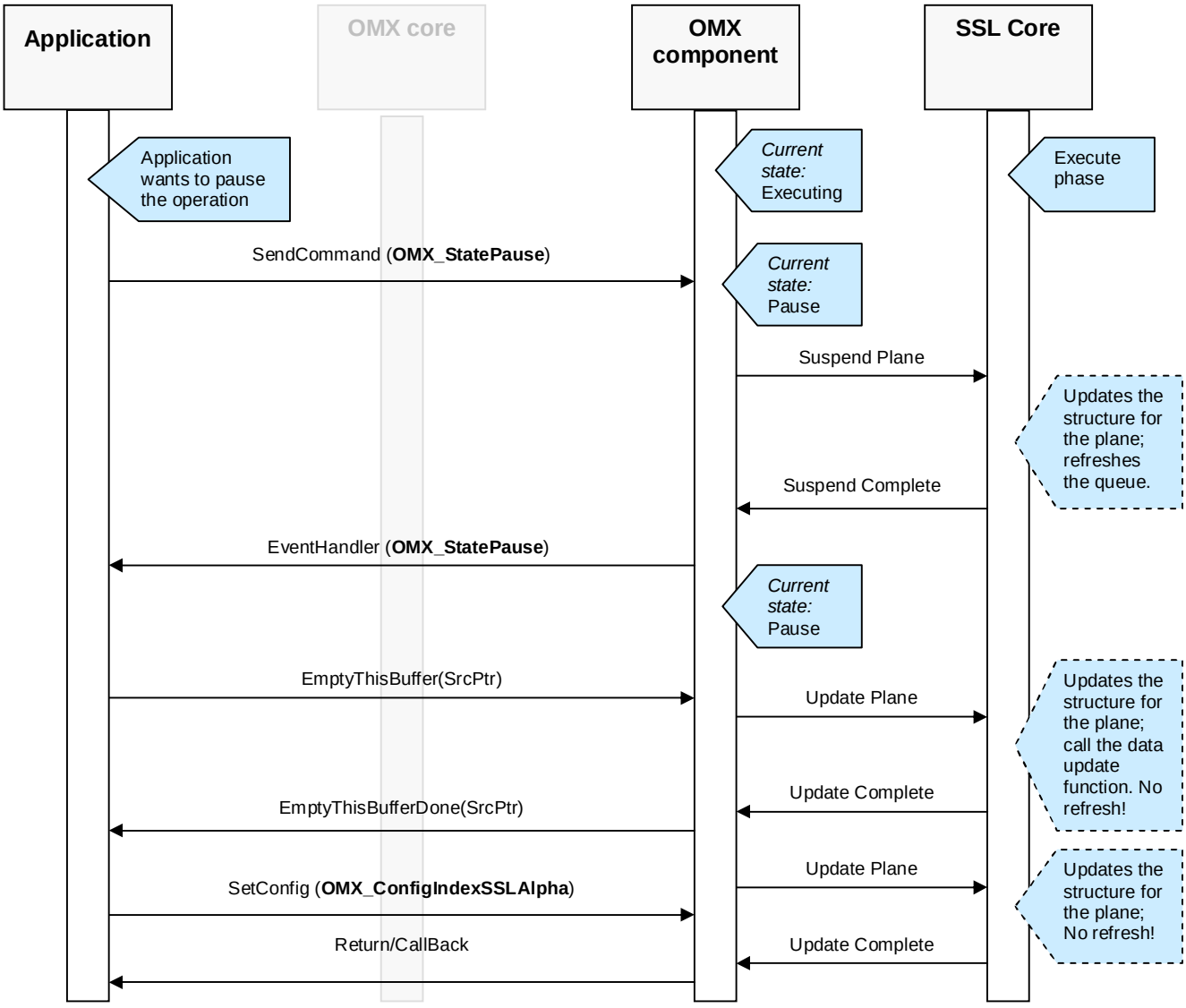
Figure 13 OMX component de-initialization: Plane Destroy process

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

6.2.7 Component Pause (Transition Executing à Pause)

Prior to pausing the component the must be in the executing state.

- The Application calls SendCommand with OMX_StatePause.
- The component will set its state to Pause.
- The component makes the current plane suspended. This would mean that the SSL Core would skip this plane in the scan for refresh. The SSL Core updates the plane structure with this information. It however does not remove it from the queue. It then refreshes the framebuffer.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.
- While in the Pause state any input buffers sent by Application will cause the SSL Component to instruct the SSL Core to update the shadowbuffer, but the plane shall be skipped in the refresh cycle. If a content update has happened for a paused buffer, the refresh cycle is not called. Any configuration parameter update would e reflected by the SSL in the plane structure, but no refresh shall be called.



PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Figure 14 OMX component execution: SSL Pausing

6.2.8 Component Resume (Transition Pause à Executing)

Prior to resuming the component the must be in the pause state.

- The Application calls SendCommand with OMX_StateExecuting.
- The component will set its state to Executing.
- The OMX SSL Component instructs the SSL Core to change the state of the plane from suspended to normal.
- The SSL Core also calls the refresh cycle with the latest set of paramers available and acknowledges the SSL Client.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.

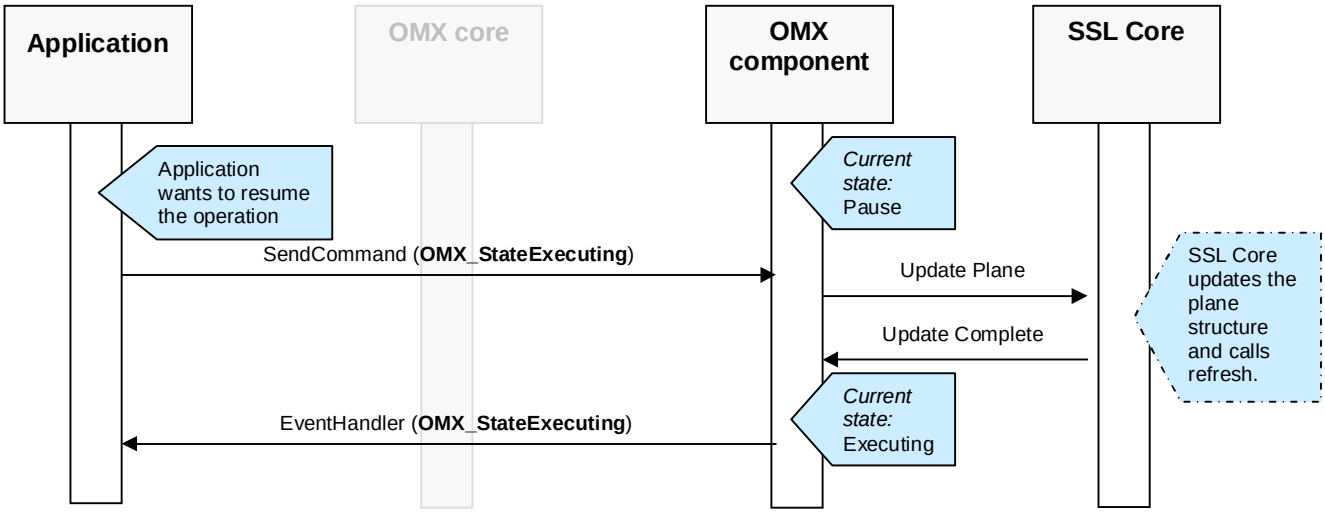


Figure 15 OMX Component Pause: Return to Execute state

6.2.9 Component Stop (Transition Pause à Idle)

From the pause state, it is possible for the component to be returned to the idle state.

- Application issues SendCommand with OMX_StateIdle.
- If the SSL Client is in Pause state, it need not wait for an update from the SSL. It sends the command to the SSL Core to deactivate.
- Once the SSL Core acknowledges the deactivate call, the SSL Client moves the Idle State.
- EventHandler application callback is called to notify Application of a successful state transition or in case there was an error.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

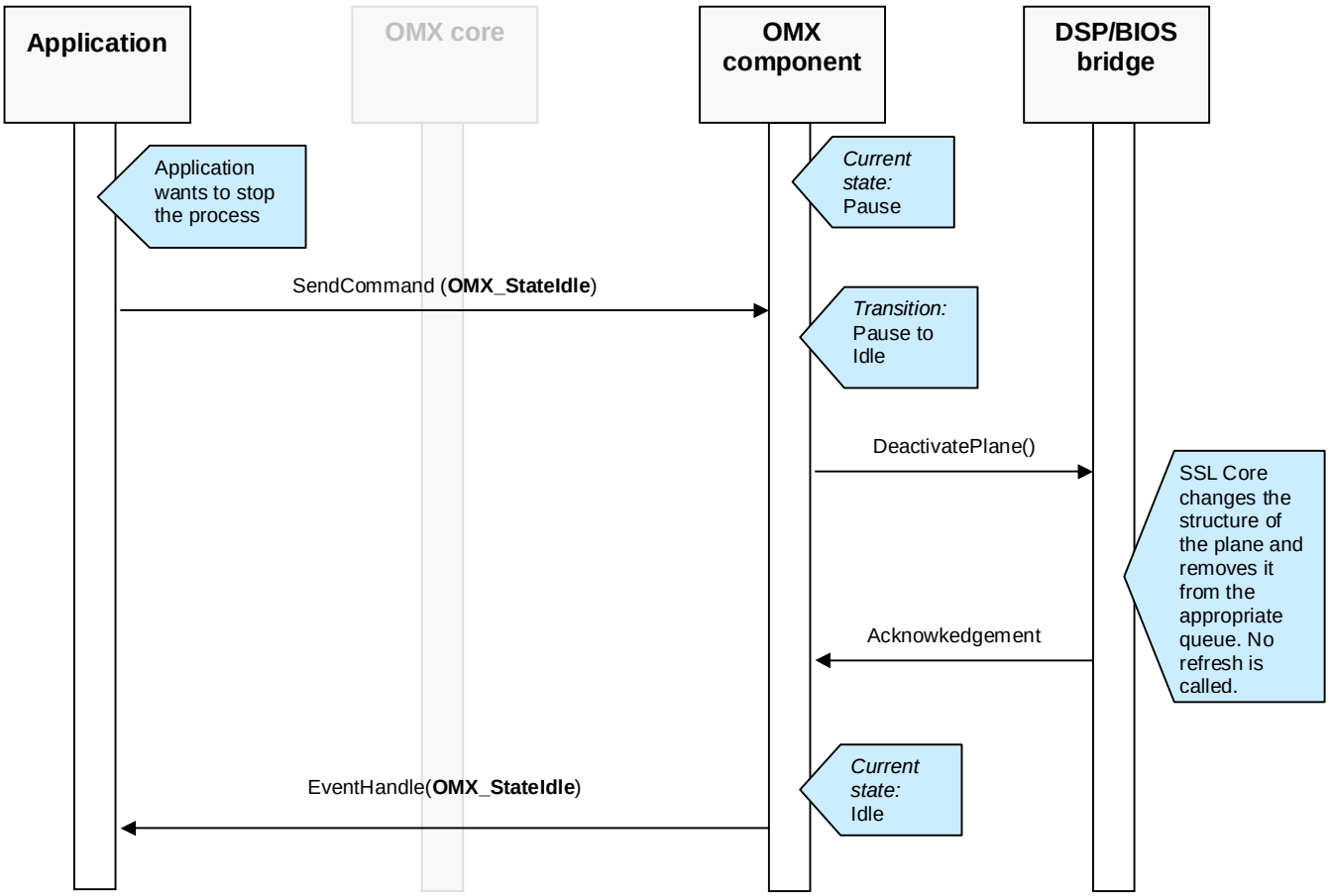


Figure 16 OMX Component Pause: Return to Idle state

6.3 Input and Output Buffer Allocation Scenarios

The input buffer shall always be allocated by the Application. It shall however configure the SSL Client input port with the plane dimensions for the SSL to allocate its Shadow Buffer. The only exception to this is the case where a DSA is requested. In this case, the Framebuffer shall be provided as the plane. The Application shall use the Framebuffer region it has requested to directly write the input. Even after doing this, the Application needs to call the EmptyThisBuffer() function. This causes the SSL Component to instruct the SSL Core to update the Display. DSA Mode is explained in detail in Appendix.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

7 Software Requirements

This section needs to filled with the requirements table.

8 Requirements Traceability

This section describes the Data types, data structures, callbacks and macros, which are supported by the OpenMAX™ 1.0.

8.1 Defined Types

8.1.1 OMX Basic Data Types

OpenMAX™ 1.0 Core exports set of data types which are given in OpenMAX™ 1.0 Core Design Specification. OpenMAX™ 1.0 Core uses these data types instead of fundamental C-types like int; char etc. in order to provide portability across different platforms, compilers and operating systems. These data types are defined in OMX_Types.h file. The **OMX SSL Client** uses these basic data types and does not define any other basic types of its own.

8.1.2 SSL Core Data Types

The SSL Core defines its own data types, which must be used in invoking the SSL Core features. The **OMX SSL Client** uses these data types to communicate with the SSL Core. The **OMX SSL Client** also translates data types passed by the Application to the types understood by the SSL Core.

8.1.2.1 OMX_SSL_DISPLAY_ORIENTATION_ENUM

This is an enumeration that is used to set the orientation of the display.

```
typedef enum{
        ORIENTATION_PORTRAIT = 0,
        ORIENTATION_LANDSCAPE
        } OMX_SSL_DISPLAY_ORIENTATION_ENUM;
```

8.1.2.2 OMX_CONFIG_SSLPLANE_APLHA_ENUM

This is an enumeration that lists the currently available alpha values for plane blending. This is in agreement with the feature set supported by the IMG Layer with respect to Alpha Blending. This type oif data can be used to set/get the alpha parameter configuration for the current plane.

8.1.2.3 OMX_CONFIG_DISPLAY_FORMATSSUPPORTED

This is an unsigned integer of type OMX_U32. The position and value of bits indicates the presence/absence of support for the given formats.

Bit Position	Format (0: UnSupported, 1: Supported)
0	OMX_COLOR_FormatMonochrome
1	OMX_COLOR_Format8bitRGB332
2	OMX_COLOR_Format12bitRGB444
3	OMX_COLOR_Format16bitARGB4444
4	OMX_COLOR_Format16bitARGB1555

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Bit Position	Format (0: UnSupported, 1: Supported)
5	OMX_COLOR_Format16bitRGB565
6	OMX_COLOR_Format16bitBGR565
7	OMX_COLOR_Format18bitRGB666
8	OMX_COLOR_Format18bitARGB1665
9	OMX_COLOR_Format19bitARGB1666
10	OMX_COLOR_Format24bitRGB888
11	OMX_COLOR_Format24bitBGR888
12	OMX_COLOR_Format24bitARGB1887
13	OMX_COLOR_Format25bitARGB1888
14	OMX_COLOR_Format32bitBGRA8888
15	OMX_COLOR_Format32bitARGB8888
16	OMX_COLOR_FormatYUV411Planar
17	OMX_COLOR_FormatYUV411PackedPlanar
18	OMX_COLOR_FormatYUV420Planar
19	OMX_COLOR_FormatYUV420PackedPlanar
20	OMX_COLOR_FormatYUV420SemiPlanar
21	OMX_COLOR_FormatYUV422Planar
22	OMX_COLOR_FormatYUV422PackedPlanar
23	OMX_COLOR_FormatYUV422SemiPlanar
24	OMX_COLOR_FormatYCbYCr
25	OMX_COLOR_FormatYCrYCb
26	OMX_COLOR_FormatCbYCrY
27	OMX_COLOR_FormatCrYCbY
28	OMX_COLOR_FormatYUV444Interleaved
29	OMX_COLOR_FormatRawBayer8bit
30	OMX_COLOR_FormatRawBayer10bit
31	OMX_COLOR_FormatRawBayer8bitcompressed

Table 6 Display Formats Supported and Bit position for query

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

8.2 Data Structures

8.2.1 Component private data structure

The OMX SSL Client maintains all the data required for its operation in this structure. This structure cannot be accessed by the Application.

Table 7 OMX_SSL_COMPONENT_PRIVATE_DATA structure

Data field Name	Description
OMX_BUFFERHEADERTYPE *buffer_headers[OMX_SSL_N_PORTS]	Each element in this array is a pointer to an array of buffer headers. Each array of buffer headers is associated with one port (input port or output port) of the component. Each buffer header contains information about the size of the buffer and its address.
OMX_COMPONENTTYPE *componentHandle	A copy of the component's handle as returned by GetHandle is maintained in the private data area
OMX_STATETYPE currentState	Maintains the current state of the **OMX SSL Client**.
OMX_STATETYPE nextState	This is required when **OMX SSL Client** cannot make a state transition immediately. For example, when transitioning from Execute to Idle state or when transitioning from Pause to Idle state, the component must wait until the owner has reclaimed all buffers.
OMX_VIDEO_PORTDEFINITIONTYPE portDefinitions [OMX_SSL_N_PORTS]	An OMX structure which defines the ports. This includes both the input and output port.
OMX_SSL_PLANE_HANDLE_TYPE planeHandle	An OMX structure which defines the plane details.
OMX_SSL_PLANE_CONFIG planeConfiguration	A structure that the SSL Client maintains for the plane configuration.
OMX_CALLBACKTYPE applicationCallbacks	An OMX structure that stores callbacks provided by Application.

8.2.2 SSL Plane Query structure

This structure is used to convey the current plane configuration to the Application from the OMX SSL Client.

Table 8 OMX_SSL_PLANE_QUERY instance structure

Data field Name	Description
OMX_BYTE SourceDataPtr	Address provided by the application for this plane.
OMX_BYTE SSLShadowPtr	Address of the shadow buffer allocated by the SSL Core for this plane.
OMX_U32 XOffset	X Axis Offset for this plane in the Framebuffer with (0,0) as the top left.
OMX_U32 XOffset	Y Axis Offset for this plane in the Framebuffer with (0,0) as the top left.
OMX_U32 XLen	Width of the plane.
OMX_U32 YLen	Height of the plane.
OMX_CONFIG_SSLPLANE_APLHA_ENUM Alpha	AlphaBlending Parameter for this plane.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Data field Name	Description
OMX_BOOL bActive	Flag that indicates if the plane is active; default=0 = inactive.
OMX_BOOL bAlwaysOnTop	Flag that indicate if this is an 'Always On Top' Plane.
OMX_BOOL bSuspended	Flag that is set if the OMX Component goes to Pause. Suspended Buffers have their configuration updated, retain their place in the queue, but not considered for a refresh.
OMX_BOOL bDSAPlane	Flag that indicates if this plane is a DSA plane. If this is the case, the SSLShadowPtr and the FrameBuffPtr below should be identical.
OMX_BYTE SSLFrameBufferPtr	The framebuffer pointer. In case DSA is used, this field is identical to the SSLShadowPtr.
OMX_BOOL bIsDSAPlane	This flag indicates if this plane is a DSA Region Plane. Refer to Appendix for info on DSA.

8.2.3 SSL Plane Configuration Structure

This structure is used to configure the current plane. It is used by the Applicatio with a SetConfig call.

Table 9 OMX_SSL_PLANE_CONFIG instance structure

Data field Name	Description
OMX_U32 XOffset	X Axis Offset for this plane in the Framebuffer with (0,0) as the top left.
OMX_U32 YOffset	Y Axis Offset for this plane in the Framebuffer with (0,0) as the top left.
OMX_U32 XLen	Width of the plane.
OMX_U32 YLen	Height of the plane.
OMX_CONFIG_SSLPLANE_APLHA_ENUM Alpha	AlphaBlending Parameter for this plane.
OMX_BOOL bAlwaysOnTop	Flag that indicate if this is an 'Always On Top' Plane.
OMX_BOOL bDSAPlane	Flag that makes this plane region accessible using DSA. Refer to Appendix for more information.

8.2.4 SSL Plane Handle Structure

This structure is defined in the SSL_Core.h and is used by the SSL OMX Component to provide plane configuration parameters. The handle is provided by the SSL Core after a CreatePlane is performed. The handle merely contains a pointer to the structure allocated by the SSL Core for the plane.

Table 10 OMX_SSL_PLANE_HANDLE_TYPE structure

Data field Name	Description	Default value
OMX_SSL_PLANE_CONFIG * sslPlaneConfiguration	The SSL OMX Component obtains the pointer to the Structure allocated by the SSL Core for the plane in this parameter.	NULL

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

8.2.5 SSL Display Query Structure

This structure is used to Query a Physical Display for its parameters. This structure is used with the GetParameter function of the SSL Client.

Table 11 OMX_SSL_DISPLAY_QUERY_TYPE structure

Data field Name	Description
OMX_U32 DisplayMaxWidth	This is the maximum display width for the physical display.
OMX_U32 DisplayMaxHeight	This is the maximum display height for the physical display.
OMX_CONFIG_DISPLAY_FORMATSSUPPORTED DisplayDataFormats	The display formats supported. This is a bitmap – bit position and value indicate is a format is supported or not.
OMX_BOOL DisplayDitherSupport	Indicates if dithering is supported or not.
OMX_SSL_DISPLAY_ORIENTATION_ENUM DisplayOrientation	This is used to indicate the current orientation for display.
OMX_BOOL DisplayEnabled	This is a parameter to enable the physical display.
OMX_U32 DisplayDeviceID	This is the hardware device id.
OMX_CONFIG_BACKLIGHTTYPE DisplayBacklightConfig	This is a structure that returns the current backlight configuration.
OMX_STRING VendorName	This indicates the vendor name for the display.
OMX_BOOL bIsActive	Indicates if this display is active or not for the current plane.

8.2.6 SSL Display Configuration Structure

This structure is used to configure a Physical Disply using the SetParameter function of the SSL OMX Client.

Table 12 OMX_SSL_DISPLAY_CONFIG_TYPE structure

Data field Name	Description
OMX_COLOR_FORMATTYPE DisplayFormat	This is used to set the display format for the current physical display.
OMX_BOOL DitherEnable	This is used to enadle dithering in the display.
OMX_SSL_DISPLAY_ORIENTATION_ENUM Orientation	This parameter is used to set the orientation for the currently set physical display.
OMX_BOOL bEnable	This is used to switch ON/OFF the physical display.
OMX_CONFIG_BACKLIGHTTYPE DisplayBackLightConfig	This is used to configure that backlight configuration.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

8.2.7 SSL Display Dvices Listing Structure

This structure returns the number of devices available, the device id and vendor name for each of the devices.

Table 13 OMX_SSL_AVAILABLEDISP_TYPE structure

Name	Type	Description
nDisplays	OMX_U32	This parameter gives the number of display devices available.
VendorNames	OMX_STRING	This parameter gives the names of the vendors separated by a ‘,’. This is a null terminated string. There are nDisplay Vendor Names.
DeviceIDs	OMX_STRING	This parameter gives the Device IDs of the display devices available separated by ‘,’. This is a Null terminated string. There are nDisplays devices ids.

8.3 API Requirements Coverage

The OpenMAX™ 1.0 core provides a set of macros that are used by Application to perform various operations like loading the component, communicating with OMX component etc. These macros are defined in OMX_Core.h. Each macro maps to a function implemented by the OMX component. Detailed description of each function implemented by the OMX SSL Component and is given in following sub sections. Application (BMI/MMI) must not call any of these functions directly and instead use the macros provided by the OMX core.

8.3.1 Common pre conditions

Before BMI/MMI can invoke the functions of theOMX SSL Component, the component must be loaded. BMI/MMI loads the component by calling OMX_GetHandle, which is an OMX core function. For details on component loading and unloading refer to OpenMAX™ 1.0 Core Design Specification.

8.3.2 OMX_SSL_ComponentInit

```
OMX_ERRORTYPE OMX_SSL_ComponentInit (OMX_HANDLETYPE hComp)
```

Description

This function is called by OMX core. BMI/MMI needs to call OMX_GetHandle, as described in OpenMAX™ 1.0 Core Design Specification. The OMX core maintains a table that lists all the OMX components and their ComponentInit functions. The OMX_SSL_ComponentInit function must be present in this component table.

Parameters

Name	Type	Description
hComp	OUT	The component fills the handle structure with pointers to functions that it implements. After this, BMI/MMI can use these function pointers to access the functionality of this OMX component

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned. For a description of error codes, refer OpenMAX™ 1.0 Core Design Specification.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function.
- Entry function to the OMX component.
- Populates hComp with function pointers to functions implemented by this particular OMX component.
- Allocates private data of the OMX component.
- Sets the current state of the OMX component as OMX_StateLoaded.

8.3.3 OMX_SSL_SetCallbacks

```
OMX_ERRORTYPE OMX_SSL_SetCallbacks (OMX_HANDLETYPE hComp,
                                    OMX_CALLBACKTYPE* pCallBacks,
                                    OMX_PTR pAppData)
```

Description

This function is called by OMX core. BMI/MMI needs to call OMX_GetHandle, as described in OpenMAX™ 1.0 Core Design Specification. After calling ComponentInit, the OMX core calls this function to provide application callbacks to the OMX component.

Parameters

Name	Type	Description
hComp	IN	Handle of the component to be accessed. This is the component handle returned by the call to the GetHandle function
pCallBacks	IN	Pointer to an OMX_CALLBACKTYPE structure used to provide the callback information to the component
pAppData	IN	Pointer to an application defined value. It is anticipated that the application will pass a pointer to a data structure or a “this pointer” in this area to allow the callback (in the application) to determine the context of the call

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned. For a description of error codes, refer OpenMAX™ 1.0 Core Design Specification.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function.
- Copies the contents of the callback structure into the private data area for later use.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

8.3.4 OMX_SSL_GetComponentVersion

```
OMX_ERRORTYPE OMX_SSL_GetComponentVersion (OMX_HANDLETYPE hComp,
                                          OMX_STRING* pComponentName,
                                          OMX_VERSIONTYPE* pComponentVersion,
                                          OMX_VERSIONTYPE* pSpecVersion,
                                          OMX_UUIDTYPE* pComponentUUID)
```

Description

Queries the component and returns information about the component. BMI/MMI calls this function to get the version of the component.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
PComponentName	OUT	This holds the name of the component at the successful return from this macro. The maximum length of the name is 128 including the null terminating character.
PComponentVersion	OUT	This is a pointer to the OMX_VERSIONTYPE structure which is filled by the component. The component fills the component version information in this structure
pSpecVersion	OUT	This is a pointer to the OMX_VERSIONTYPE structure which is filled by the component. The component fills the OMX specification version information in this structure.
PComponentUUID	OUT	This is a pointer to the DSP_UUID structure which is be filled by the component.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function.
- Fills component version and OMX specification version in the structures passed by BMI/MMI.

8.3.5 OMX_SSL_SendCommand

```
OMX_ERRORTYPE OMX_SSL_SendCommand (OMX_HANDLETYPE hComp,
                                   OMX_COMMANDTYPE Cmd,
                                   OMX_U32 nParam)
```

Description

Sends a command to the component.

Parameters

Name	Type	Description
hComp	IN	This input argument is component handle.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Name	Type	Description
Cmd	IN	This specifies the command type/category. The value of this argument can be OMX_CommandStateSet. Currently in OpenMAX™ 1.0, the only valid command is to set the component state.
NParam	IN	The value of this argument is dependent on the cmd argument. If the value of the cmd is OMX_CommandStateSet, this argument contains the state that is to be set for the component. The value can be one of the values defined by OMX_STATETYPE.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Non-blocking function.
- Handles state transitions as described in sections 6.1 and 6.2.
- Since the OMX component is executed in the same task as that of the BMI/MMI, SendCommand will execute the command in-context.
- Calls ProcessFunction to handle input and output buffers. See details in section 8.6.
- For successful state transitions, EventHandler callback would be invoked.
- If new state specified is the same as the current state of the OMX component, no actual state change processing will occur. The EventHandler callback is not called. Instead the component will only check if there are any buffers waiting on the input and output data pipes and handle these as required.

8.3.6 OMX_SSL_GetParameter

```
OMX_ERRORTYPE OMX_SSL_GetParameter (OMX_HANDLETYPE hComp,
                                    OMX_INDEXTYPE nParamIndex,
                                    OMX_PTR pCompParam)
```

Description

Gets the current parameter settings of the component. BMI/MMI should allocate memory for the correct structure and pass it to this function.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
NParamIndex	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
pCompParam	OUT	This is a pointer to the structure which needs to be filled in by the component.

Supported nParamIndex values and their corresponding structures are:

nParamIndex	DataType	Include file
OMX_IndexParamSSLInputPort	OMX_SSL_PORTDEFINITIONTYPE	OMX_SSL_component.h
OMX_IndexParamAvailableDisplays	OMX_SSL_AVAILABLEDISP_TYPE	OMX_SSL_component.h
OMX_IndexParamDisplayProperties	OMX_SSL_DISPLAY_QUERY_TYPE	OMX_SSL_component.h
OMX_IndexParamDisplayMaxWidth	OMX_U32	OMX_Types.h

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

OMX_IndexParamDisplayMaxHeight	OMX_U32	OMX_Types.h
OMX_IndexParamDisplayDataFormat	OMX_CONFIG_DISPLAY_FORMATSSUPPORTED	OMX_SSL_component.h
OMX_IndexParamDisplayDitherSupport	OMX_BOOL	OMX_Types.h
OMX_IndexParamDisplayOrientation	OMX_SSL_DISPLAY_ORIENTATION_ENUM	OMX_SSL_component.h
OMX_IndexParamDisplayEnabled	OMX_SSL_DISPLAY_ENABLE	OMX_SSL_component.h
OMX_IndexParamDiplayBackLitConfig	OMX_CONFIG_BACKLIGHTTYPE	OMX_IVCommon.h
OMX_IndexParamDisplayDeviceID	OMX_U32	OMX_Types.h
OMX_IndexParamDisplayVendorName	OMX_STRING	OMX_Types.h
OMX_IndexParamDisplayIsActive	OMX_BOOL	OMX_Types.h

For details on how each of these parameter structures are mapped to what is required by the Application, refer to section 8.2.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Pre Condition

The structure specified by the third argument must have its structure size and version information filled in before invoking the macro.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function
- Copies the appropriate values from the component's private data area and populates the structure passed in the third argument
- Performs basic parameter checking by comparing the size passed in the structure (third argument to this function) to the actual size of the structure.

8.3.7 OMX_SSL_SetParameter

```
OMX_ERRORTYPE OMX_SSL_SetParameter (OMX_HANDLETYPE hComp,
                                    OMX_INDEXTYPE nParamIndex,
                                    OMX_PTR pCompParam)
```

Description

Sets the various parameters of the component with desired values. BMI/MMI should allocate memory for the correct structure, fill it with the required values and pass it to this function. The component makes a local copy of this structure and uses it at the time of initialization. This function should be used to set the initialization parameters of the component, when the component is in the LOADED state.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
NParamIndex	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
pCompParam	IN	This input argument is a pointer to a structure which the component uses to make its local copy.

Supported nParamIndex values and their corresponding structures are:

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

nParamIndex	Structure	Include file
OMX_IndexParamSSLInputPort	OMX_SSL_PORTDEFINITIONTYPE	OMX_SSL_component.h
OMX_IndexParamActiveDisplay	OMX_BOOL	OMX_Types.h
OMX_IndexParamDisplayProperties	OMX_SSL_DISPLAY_CONFIG_TYPE	OMX_SSL_component.h
OMX_IndexParamDisplayDataFormat	OMX_COLOR_FORMATTYPE	OMX_IVCommon.h
OMX_IndexParamDisplayDitherSupport	OMX_BOOL	OMX_Types.h
OMX_IndexParamDisplayOrientation	OMX_SSL_DISPLAY_ORIENTATION_ENUM	OMX_SSL_component.h
OMX_IndexParamDisplayEnabled	OMX_BOOL	OMX_Types.h
OMX_IndexParamDisplaySetActive	OMX_BOOL	OMX_Types.h

For details on how each of these parameter structures are mapped to what is required by the Application, refer to section 8.3.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function.
- Copies the structure passed into the component's private data area for later use.
- Performs basic parameter checking by comparing the size passed in the structure (third argument to this function) to the actual size of the structure.

8.3.8 OMX_SSL_GetConfig

```
OMX_ERRORTYPE OMX_SSL_GetConfig (OMX_HANDLETYPE hComp,
                                 OMX_INDEXTYPE nConfigIndex,
                                 OMX_PTR pCompConfig)
```

Description

Gets the configuration parameters of the component. This function can be invoked at any time after the component has been loaded. BMI/MMI allocates the required structure and passes it to this function. The component fills this structure with the required information.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
nConfigIndex	IN	This identifies the structure being used by the third argument. Values are defined in OMX_index.h
pCompConfig	OUT	This output argument is the pointer to the structure to be filled by the component.

Supported nConfigIndex values and their corresponding structures are:

nConfigIndex	Type	Include file
OMX_IndexConfigSSLPlaneAlpha	OMX_CONFIG_SSLPLANE_APLHA_ENUM	OMX_SSL_component.h
OMX_IndexConfigSSLPlaneAlwaysOnTop	OMX_BOOL	OMX_Types.h
OMX_IndexConfigSSLPlaneActiveFlag	OMX_BOOL	OMX_Types.h
OMX_IndexConfigSSLPlaneSuspendedFlag	OMX_BOOL	OMX_Types.h
OMX_IndexConfigSSLPlaneSrcPtr	OMX_BYTE	OMX_Types.h
OMX_IndexConfigSSLPlaneShadowPtr	OMX_BYTE	OMX_Types.h

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

OMX_IndexConfigSSLPlaneXOffset	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneYOffset	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneXLen	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneYLen	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneFrameBuffPtr	OMX_BYTE	OMX_Types.h
OMX_IndexConfigSLLPlaneIsDSAPlane	OMX_BOOL	OMX_Types.h
OMX_IndexConfigSSLPlaneConfig	OMX_SSL_PLANE_QUERY	OMX_SSL_component.h

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Pre Condition

The structure specified by the third argument must have its structure size and version information filled in before invoking the macro. Please refer to section 8.3 for the description of these structures.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with Texas Instruments OpenMAX™ 1.0 specification.

Implementation

- Blocking function
- Copies the appropriate values from the component's private data area and populates the structure passed in the third argument
- Performs basic parameter checking by comparing the size passed in the structure (third argument to this function) to the actual size of the structure.

8.3.9 OMX_SSL_SetConfig

```
OMX_ERRORTYPE OMX_SSL_SetConfig (OMX_HANDLETYPE hComp,
                                 OMX_INDEXTYPE nConfigIndex,
                                 OMX_PTR pCompConfig)
```

Description

Sets the various parameters of the component with desired values. This function can be invoked at any time after the component has been loaded. BMI/MMI should allocate memory for the correct structure, fill it with the required values and pass it to this function. The component makes a local copy of this structure and uses it configure the codec at the appropriate moment.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
NConfigIndex	IN	This identifies the structure being used by the third argument . Values are defined in OMX_index.h
pCompConfig	OUT	This input argument is a pointer to a structure that holds the values with which codec is to be configured

Supported nConfigIndex values and their corresponding structures are:

nConfigIndex	Type	Include file
OMX_IndexConfigSSLPlaneAlpha	OMX_CONFIG_SSLPLANE_APLHA_ENUM	OMX_SSL_component.h
OMX_IndexConfigSSLPlaneAlwaysOnTop	OMX_BOOL	OMX_Types.h
OMX_IndexConfigSSLPlaneXOffset	OMX_U32	OMX_Types.h

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

OMX_IndexConfigSSLPlaneYOffset	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneXLen	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneYLen	OMX_U32	OMX_Types.h
OMX_IndexConfigSSLPlaneConfig	OMX_SSL_PLANE_CONFIG	OMX_SSL_component.h
OMX_IndexConfigSSLPlaneDSAPlane	OMX_BOOL	OMX_Types.h

For details on how each of these configuration structures is mapped to what is required by the Application, refer to section 8.3.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function.
- Copies the structure passed into the component's private data area for later use.
- Performs basic parameter checking by comparing the size passed in the structure (third argument to this function) to the actual size of the structure.

8.3.10 OMX_SSL_GetState

```
OMX_ERRORTYPE OMX_SSL_GetState (OMX_HANDLETYPE hComp,
                                OMX_STATETYPE* pState)
```

Description

The BMI/MMI calls this function to get the current state of the component.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pState	OUT	This is the output argument, which points to the memory location where the component should store its current state. This argument should not be NULL.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function
- Copies the current state value stored in the component's private data area into the structure passed to the function

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

8.3.11 OMX_SSL_EmptyThisBuffer

```
OMX_ERRORTYPE OMX_SSL_EmptyThisBuffer (OMX_HANDLETYPE hComp,
                                       OMX_U32 nPortIndex,
                                       OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

BMI/MMI uses this function to send a buffer filled with input data to the input port of the component. This function will write the buffer into the input data pipe of the component. Once the component completes reading this buffer, it will notify BMI/MMI using the callback function EmptyBufferDone.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
nPortIndex	IN	This specifies an input port of the component.
pBuffer	IN	This is a pointer to the buffer header whose buffer is to be emptied.

Return

If the command successfully executes, the return code will be OMX_ErrorNone. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Stores the buffer provided by BMI/MMI into the input data pipe
- Calls SSL Core Task, which is an internal See details in section 8.5.

8.3.12 OMX_SSL_FillThisBuffer

```
OMX_ERRORTYPE OMX_SSL_FillThisBuffer (OMX_HANDLETYPE hComp,
                                      OMX_U32 nPortIndex,
                                      OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

This Function is not implemented in the current SSL Version.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
nPortIndex	IN	This specifies an output port of the component.
pBuffer	OUT	This is a pointer to the buffer header whose buffer is to be filled.

Return

This function always returns OMX_ErrorNotImplemented.

Requirement Coverage

This method addresses requirement:

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Not implemented
- Always returns OMX_ErrorNotImplemented

8.3.13 OMX_SSL_ComponentTunnelRequest

```
OMX_ERRORTYPE OMX_SSL_ComponentTunnelRequest (OMX_HANDLETYPE hComp,
                                              OMX_U32 nPortInput,
                                              OMX_HANDLETYPE hTunneledComp,
                                              OMX_U32 nTunneledPort,
                                              OMX_DIRTYPE eDir,
                                              OMX_CALLBACKTYPE* pCallbacks)
```

Description

This function is not implemented in the current design of the SSL Component.

Parameters

Name	Type	Description
HComp	IN	This input argument is the component handle.
NPortInput	IN	
hTunneledComp		
nTunneledPort		
eDir		
pCallbacks		

Return

This function always returns OMX_ErrorNotImplemented.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Not implemented
- Always returns OMX_ErrorNotImplemented

8.3.14 OMX_SSL_ComponentDeInit

```
OMX_ERRORTYPE OMX_SSL_ComponentDeInit(OMX_HANDLETYPE hComp)
```

Description

This function is called by OMX core when BMI/MMI calls OMX_FreeHandle, as described in OpenMAX™ 1.0 Core Design Specification.

Parameters

Name	Type	Description
hComp	OUT	Handle of the component to be accessed. This is the component handle returned by the call to the GetHandle function

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Return

If the command successfully executes, the return code will be OMX_NoError. Otherwise the appropriate OMX error will be returned.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Blocking function.
- Releases private data of the OMX component.
- Performs component specific de-initializations, related to the underlying codec or algorithm.

8.4 Application callbacks

The OpenMAX™ 1.0 specification requires BMI/MMI to provide three callbacks for buffer exchange and event handling. At loading time, the OMX component receives a structure containing pointers to the callback functions. The OMX component makes a copy of this structure in the private data area. This section describes the callback functions.

8.4.1 EventHandler

```
void (*EventHandler)(
OMX_HANDLETYPE hComp,
OMX_PTR pAppData,
OMX_EVENTTYPE eEvent,
OMX_U32 Data,
OMX_STRING cExtraInfo)
```

Description

The EventHandler method is used to notify BMI/MMI when an event of interest occurs. This event may be change of state, an error occurred etc.

Parameters

Name	Type	Description
hComp	IN	This input argument is the component handle.
pAppData	IN	Pointer to data, which was defined by application when the component was loaded. Using this data, the application identifies who invoked this callback.
eEvent	IN	One of the component events that are defined in OMX_EVENTTYPE enumeration. This can be state change, an error etc.
Data	IN	Used only if an error event occurs. Data will be OMX_ERRORTYPE.
cExtraInfo	IN	String, which may carry some more explanation about the error. It is not always required for a component to use this argument.

Return

None

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

Implementation

- EventHandler is called to notify state changes in the OMX component to BMI/MMI.
- In the SSL OMX Component, this callback is invoked from ProcessFunction. See details in section **Error! Reference source not found.**.

8.4.2 EmptyBufferDone

```
void (*EmptyBufferDone)(
OMX_HANDLETYPE hComp,
OMX_PTR pAppData,
OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

This is the callback function provided by BMI/MMI, which the component uses to return an empty input buffer, for BMI/MMI to write its input into. There is always a callback EmptyBufferDone from the component for each OMX_EmptyThisBuffer call from the BMI/MMI.

When BMI/MMI allocates input buffers, it provides them to the SSL Component by calling OMX_EmptyThisBuffer. Hence the first EmptyBufferDone, for each input buffer, would be called only after the component has read the input buffer.

Parameters

Name	Type	Description
hcomp	IN	This input argument is the component handle.
pAppData	IN	Pointer to the data which was defined by the application when the component was loaded. Using this data, the application identifies who invoked this callback.
pBuffer	IN	Pointer to buffer header structure which contains pointer to emptied buffer, its size etc.

Return

None

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- EmptyBufferDone is invoked when the codec has completed processing the input data and generated the output data.
- This happens when the SSL OMX Component receives the acknowledgement from the SSL Core task for display update completion.

8.4.3 FillBufferDone

```
void (*FillBufferDone)(
OMX_HANDLETYPE hComp,
OMX_PTR pAppData,
OMX_BUFFERHEADERTYPE* pBuffer)
```

Description

This is the callback function, provided by BMI/MMI, which the component uses to return a filled output buffer, for BMI/MMI to read the output from. There is always a callback FillBufferDone from the component for each call OMX_FillThisBuffer from BMI/MMI.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

This is not implemented in the SSL OMX Component as it is a Sink.

Parameters

Name	Type	Description
HComp	IN	This input argument is the component handle.
pAppData	IN	Pointer to data which was defined by the application when the component was loaded. Using this data, the application identifies who invoked this callback.
PBuffer	IN	Pointer to the buffer header structure which contains a pointer to the filled buffer, its size etc

Return

This function always returns OMX_ErrorNotImplemented.

Requirement Coverage

This method addresses requirement:

SR14062: This interface must comply with OpenMAX™ 1.0 specification.

Implementation

- Not implemented
- Always returns OMX_ErrorNotImplemented

8.5 Internal Functions

8.5.1 SSL Core Task

The SSL Core Task is a separate thread that handles multiple OMX SSL Components. Each SSL Component is associated with a plane and the SSL Core task is responsible for handling priorities as well as using the services of the IMG for the framebuffer composition. Plese refer to the CSSD SSL Core Task Design Specifications for more information on the SSL Core Task.

8.6 Non-API Requirements Coverage

This OpenMAX™ 1.0 core will comply with the TI coding guidelines, located in the Clear Case® VOB path:

`\\OMAPSW_docs\Process\Coding_Standards\OMAPSW_C_CodingStandards.doc`

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn

9 Assumptions

- BMI/MMI and the SSL Client Component run in the same task.
- The SSL Core task runs as a separate thread.
- The IMG Server is active and running and supports alpha-blending.
- The LCD Manager is active.

10 Appendix A – Direct Screen Access (DSA)

This appendix describes the DSA mode of operation for the SSL OMX Component.

10.1 Introduction to DSA

DSA or Direct Screen Access is a mechanism by which the application gets direct access to a display region. In case of SSL, this means that, the application would have direct access to the FrameBuffer. The FrameBuffer is the destination buffer from which the LCD update takes place.

In such a scheme, a data copy from the application memory region to the shadow buffer and the alpha blending with other planes are avoided. This also saves the memory that would have otherwise been allocated for a shadow buffer.

Since DSA provides direct access to the framebuffer, sufficient protection is need to prevent conflicting data writes to DSA regions. Hence, although multiple DSA planes can exist, these must map to mutually exclusive regions in the framebuffer.

Thus the primary difference between normal mode of operation and DSA mode of access is that, the SSL Core task does not do any data/parameter update for that plane. It calls the refresh cycle (No Data Update) wherein the framebuffer contents are all updated to the configured display.

10.2 DSA Usage

In order for the application to use DSA, the application needs to follow the regular plane creation approach using a SetParameter call followed by a SetConfig call after initializing the OMX Component. In addition to the regular plane configuration parameters, the application also needs to enable the DSA Flag for this plane. In order to get control of the Framebuffer, the application needs to use the GetConfig with the index corresponsing to the framebuffer pointer.

It must be remembered that, also the DSA flag has been provided as a configuration parameter, it cannot be used once the plane has been created in the DSA mode. This is because, there would be no Shadow Buffer allocated for this. The only way is to destroy the plane and create a new one in its place.

The SSL Core also permits DSA and Non-DSA planes to co-exist. In this case, the other planes need to adhere to the same rule that applies to other DSA Planes – they need to occupy a region mutually exclusive to the DSA Plane region.

The SSL Core composes the remaing planes into the remaining region of the framebuffer and refreshes the display.

PDF 文件使用 "pdfFactory Pro" 试用版本创建 www.fineprint.cn