OMAP™ SS&P DESIGN SPECIFICATION

OpenMAX™ TI 1.5Core Nucleus® OMAPV1030

Document Revision: 1.3

Issue Date: 22 September 2005

Making**Wireless**

Table of Contents

List of Figures

List of Tables

Revision History

REV	DATE	AUTHOR	NOTES
1.0	06 Jul 2005	Mahendra Kumar H V	Initial release
1.1	25 Aug 2005	Mahendra Kumar H V, Federico Reyes	Corrections and move to new Base Document
1.2	22 Sep 2005	Federico Reyes, Praveen Rao	Updated for V1030 Nucleus Implementation.
1.3	28 Sep 2005	Praveen Rao	Updated with Inspection findings

Approvals

REV	APPROVAL 1	DATE	APPROVAL 2	DATE
1.3	David Newman	29-Sep-2005	Prathibha Tammana	

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

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

1 Introduction

This document describes the design of the TI Internal OpenMAX™ TI 1.5 Core. The environment for the OpenMAX™ TI 1.5 Core is:

- Nucleus® on OMAPV1030
- OpenMAX™ TI 1.5 components.

1.1 Purpose

This document details the design specifications for OpenMAX™ TI 1.5 Core on OMAPV1030.

1.2 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.

1.3 File Path

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

```
\\OMAPSW_SysDev\OMAPV1030\Multimedia\System_Core\Docs
```

1.4 File Name

The file name of this document is OMAPSSP_V1030_OMX_Core_DesignSpec.doc.

1.5 References

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

1.6 Definitions

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

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™ TI 1.5 are used interchangeably in the document.
OMAP™	Open Multimedia Application Platform
API	Application Programming Interface
ARM	Advanced RISC Machines
BIOS	Basic Input/Output System
DSP/BIOS™	Digital Signal Processing/Basic Input/Output System
OS	Operating System
SDP	Software Development Platform

2 Architectural Overview

The OpenMAX™ TI 1.5 framework acts as an API that defines a software interface used to provide an access layer around software or hardware components in a system. The intent of the software interface defined by OpenMAX™ TI 1.5 is to take software or hardware components with disparate initialization and command methodologies and provide a software layer having a standardized command set and standardized methodology for construction and destruction of the components. Consider a system where various components such as audio codec, audio mixer, and noise reduction filter are required. Each of these components may come from different vendors and may have entirely different methods to initialize and start them. In this scenario, the role of OpenMAX™ TI 1.5 framework is to provide an interface to high level applications with a standard protocol that allows OpenMAX™ TI 1.5 compliant components from these different vendors to be used in a standardized way. One main benefit of the OpenMAX™ TI 1.5 interface is that it abstracts the details of where the codec algorithm is running, making it easy to create a simple ARM only solution for fast development time and then later substitute a complete DSP solution without requiring major changes in the application. To achieve this objective, OpenMAX™ TI 1.5 core and OpenMAX™ TI 1.5 component are introduced between the application and the codec.

Typically OpenMAX™ TI 1.5 is used by high level applications to communicate with codec's although almost any function may be wrapped in an OMX component. One example may be to wrap the power management functions in OMX component wrappers.

The application will access an OpenMAX™ TI 1.5 component's interfaces to send commands to perform functions such as initialize, start, pause etc to the underlying codec and/or Hardware accelerator device. In this usage, the OpenMAX™ TI 1.5 component is said to wrap the underlying codec and/or Hardware accelerator device. Each codec and/or Hardware accelerator device may be wrapped by a separate OpenMAX™ TI 1.5 component or a logical group of components may be wrapped as a group wherever necessary. The OpenMAX™ TI 1.5 architecture provides standardized methods to:

- Initialize/De-initialize the component.
- Get/Set configuration of the component.
- Send various commands (like start/stop/pause etc) to codec.
- Send/Receive buffers to/from component.

The OpenMAX™ TI 1.5 Core is a thin layer between an application framework and the OpenMAX™ TI 1.5 components. The OpenMAX™ TI 1.5 Core does not participate in any operations other than load (init) and unload (de-init) of the components. After the OpenMAX™ TI 1.5 component (e.g.: PCM Decoder) is created, the application calls the component through the macros provided by OpenMAX™ TI 1.5 Core.

The OpenMAX™ TI 1.5 OMX core maintains a list of components support in an "OMX Component Table". Every OpenMAX™ TI 1.5 components must have its entry in the above table which consists of the component name and function-pointer to the OMX_ComponnetInit function.

The OMX_GetHandle method will locate the component specified by the component given name in the OMX component table and invoke the component methods to fill the component handle and setup the callbacks.

Note 1. In Nucleus® implementation of the OpenMAX™ TI 1.5 Core, the term "Load" and "Unload" are used to mean "Init" and "De-Init" respectively, since in Nucleus® all the component modules are statically linked.

2.1 System Diagram

Figure 1 shows the architecture of the OpenMAX™ TI 1.5 Core as used in a complete system. This diagram shows how an application framework will communicate with the OpenMAX™ TI 1.5 Core, OpenMAX™ TI 1.5 Component and codec on the DSP.

The OpenMAX™ TI 1.5 core is OS specific code that has the functionality necessary to locate and then load (initialize) an OpenMAX™ TI 1.5 Component into main memory for execution. The core is also responsible for unload (de-initialize) the component from memory when the application indicates that the component is no longer needed. The Core acts as a thin layer between an application and the component. The OpenMAX™ TI 1.5 Core is responsible for

- Initialization of the components
- De-Initialization of the components
- Providing APIs and macros to application so that application can interact with the component.

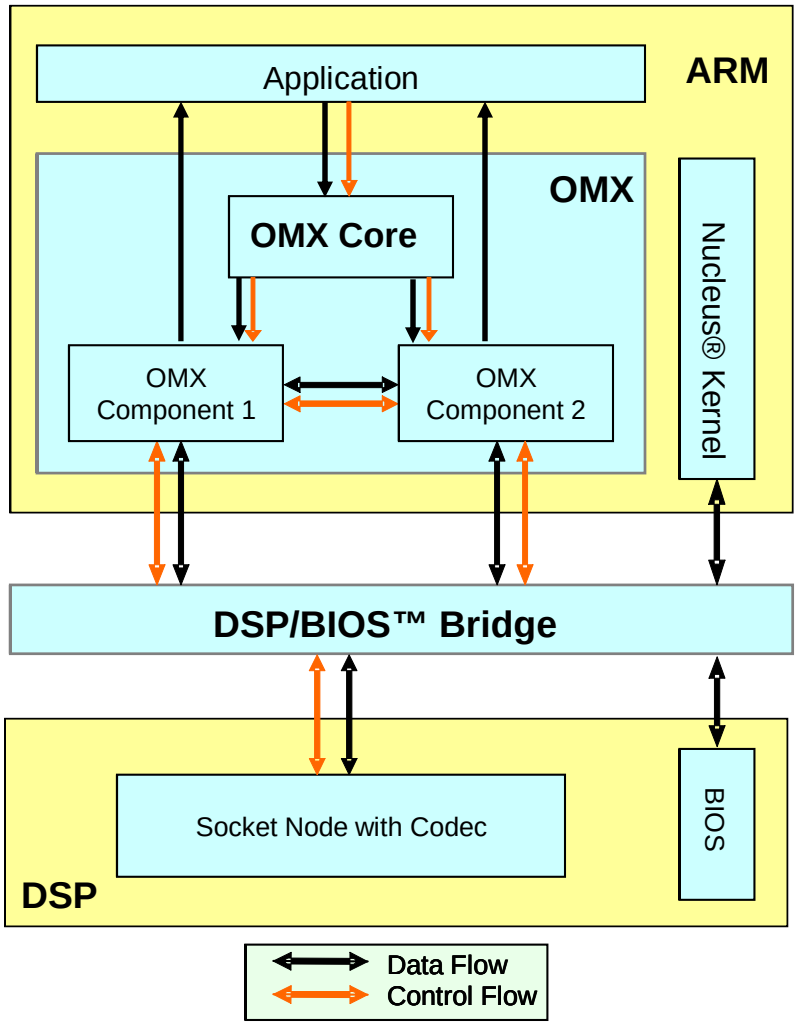


Figure 1 System Architectural Diagram

2.2 Software Design Interfaces

This section explains the interfaces that the application designer will have to use to initialize and control the OMX components in a system. The application will use the OMX core API interface to locate the component in the component table and initialize a handle for the component. Next, the application will use macros provided by the core to directly access methods within the component to setup the initialization parameters, initiate state transitions of the component and finally to pass buffers to and from the component.

Note: Refer section 9 for the interface details.

3 Design Rational

OpenMAX™ TI 1.5 is used for the development of components for OMAPV1030. OpenMAX™ TI 1.5 is a subset of the currently-in-work Khronos 1.0 standard. The TI Internal OpenMAX™ TI 1.5 standard is intended as a standard way to exchange setup information and data buffers between application and components. Since OMX core gives the standard interface to the application, an application can be used to control multiple components with minimal changes.

3.1 Relevant Specifications

Refer to the following specifications for additional information:

- Khronos OpenMAX IL Layer Specification 1.0 (in work, as of 3/3/2005)

3.2 Design Trade-offs

There are no design trade-offs specific to the core functions. Instead of providing separate APIs for state transitions, a unique macro SendCommand is provided by OMX core so that application becomes more maintainable and cleaner.

3.3 Hardware Dependencies

There are no specific hardware dependencies.

3.4 Other Pertinent Design Issues

There are no design issues currently.

4 Memory Requirements

There are no defined memory usage requirements for the OMX core such as maximum memory used. However, since the core consists of a collection of functions intended to be called by an application, the following requirements are imposed.

4.1 Memory Allocation

OMX core will not allocate memory for any of the structures of the component. The core will allocate memory only for component handle which stores function pointers of the component. The memory allocated for the component handle is freed by the core when the component is un-initialized using OMX_FreeHandle.

5 Sub-Components

The OpenMAX™ TI 1.5 Core consists of the following files each having specific functions.

5.1 OpenMAX™ TI 1.5 Core Include Files

An application includes the following OpenMAX™ TI 1.5 core files.

Table 2 OMX Core1.5 files

Core File	Function
OMX_Core.h	This file contains the exports for OpenMAX™ TI 1.5 core APIs and macros to access component APIs.
OMX_Types.h	This file contains the OpenMAX™ TI 1.5 type definitions universally followed in OpenMAX™ TI 1.5 components and OpenMAX™ TI 1.5 core.
OMX_Index.h	This file contains the index enumerations used in function calls of SetParameter / GetParameter and SetConfig / GetConfig. These enumerations are used in application and components, which provide sync mechanism to set or get parameters and do the configuration.

5.2 OpenMAX™ TI 1.5 Component Include Files

The core includes following header file which is generic to all the components.

Table 3 Component Include Files

Include files	Description or Evaluation
OMX_Component.h	This file contains definition of component handle structure i.e. OMX_COMPONENTTYPE.
OMX_ComponentTable.h	This file contains definition of the component table entry structure i.e. OMX_COMPONENTLIST

5.3 Multimedia Support Include Files

The OpenMAX™ TI 1.5 provides following header files for multimedia components support.

Table 4 Multimedia Support Include Files

Include files	Description or Evaluation
OMX_Audio.h	This file contains the structures needed by Audio components to exchange parameters and configuration data between the application and the component.
OMX_Video.h	This file contains the structures needed by Video components to exchange parameters and configuration data between the application and the component.
OMX_Image.h	This file contains the structures needed by Image components to exchange parameters and configuration data between the application and the component.
OMX_IVCommon.h	This file contains the structures needed by image and video components to exchange parameters and configuration data between the application and the component.

Include files	Description or Evaluation
OMX_Other.h	This file contains the structures needed by the other types of component to exchange parameters and configuration data between the application and the component.

6 Data Flow

The OpenMAX™ TI 1.5 core does not participate in data flow.

7 Control Flow

This section describes how control flows between the application, the OMX core and the OMX component. The various states of the component are detailed in section 7.2.

7.1 Component Phases

There are various phases in a component life cycle. The OMX core provides its own macros and APIs for each phase. The following sub sections describe each phase of the OMX component and which APIs or macros will be used in the corresponding phase.

7.1.1 Component Load

Figure 2 shows the loading phase of a component life cycle.

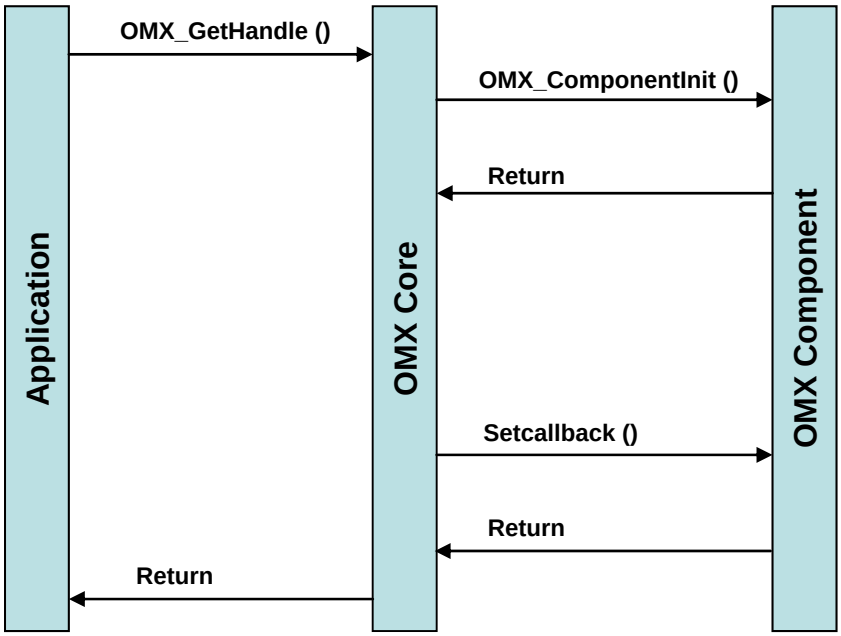


Figure 2 Load Phase of Component

This is very first phase of the component life cycle. The entry criterion for this phase is that the component must have been built as part of the Nucleus Image and the component's entry in the component table maintained by the OMX core.

The Following steps describe the process of loading the component.

- The Application calls the function OMX_GetHandle of OMX core and supplies the name of the component to be loaded and callbacks of the application as arguments to this function.
- The OMX core searches for the application specified component name in the component table. If the component entry is found, the OMX core allocates memory for the component handle and calls the registered OMX_ComponentInit function of the component. OMX_ComponentInit will fill in the component handle with the function pointers of the exported functions. Besides filling component handle with valid function pointers, the function OMX_ComponentInit also does following:
  - Checks for the availability of resources (if they are needed for component execution) and then allocates internal data structures of the component. An error may be returned if the hardware resources are not available at this time.
  - Allocates its private data area used to store the initialization parameters and set the initial values of this area to nominal values.

- OMX Core calls the function SetCallbacks of the component and supplies the application's callback function pointers to the component which is recorded in the component's private data area.

7.1.2 Component Unload

Figure 3 shows the component life cycle during the unloading stage.

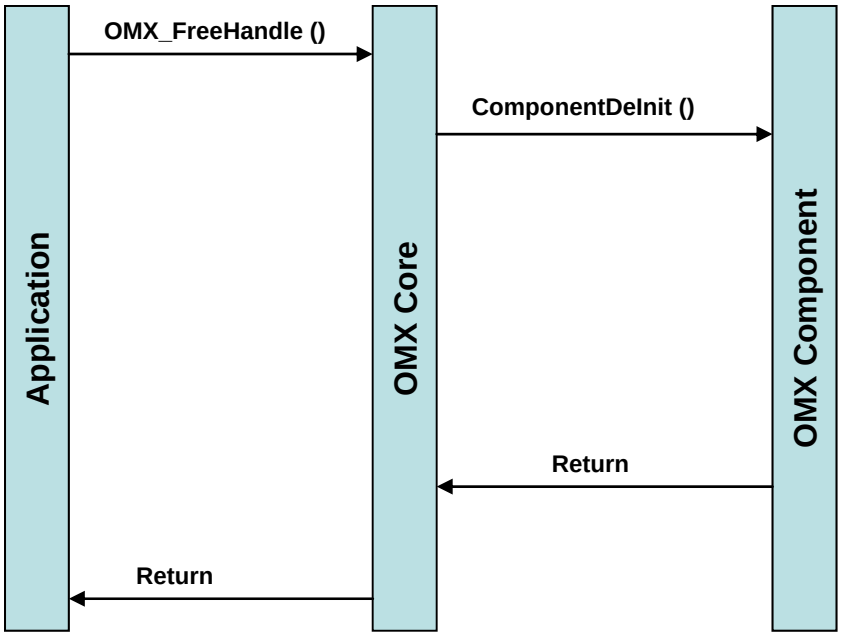


Figure 3 Unloading of Component

The application should call OMX core function OMX_FreeHandle to free the component handle which was obtained by calling OMX_GetHandle at the time of loading the component. The Core method OMX_FreeHandle in turn calls the ComponentDeInit function of the component. The OMX core then frees the memory used by the component handle. Note that after successful return from OMX_FreeHandle, the component handle is no longer valid for use by the application. On successful execution of the OMX_FreeHandle function, the component handle will be de-allocated.

7.1.3 Component Initialization

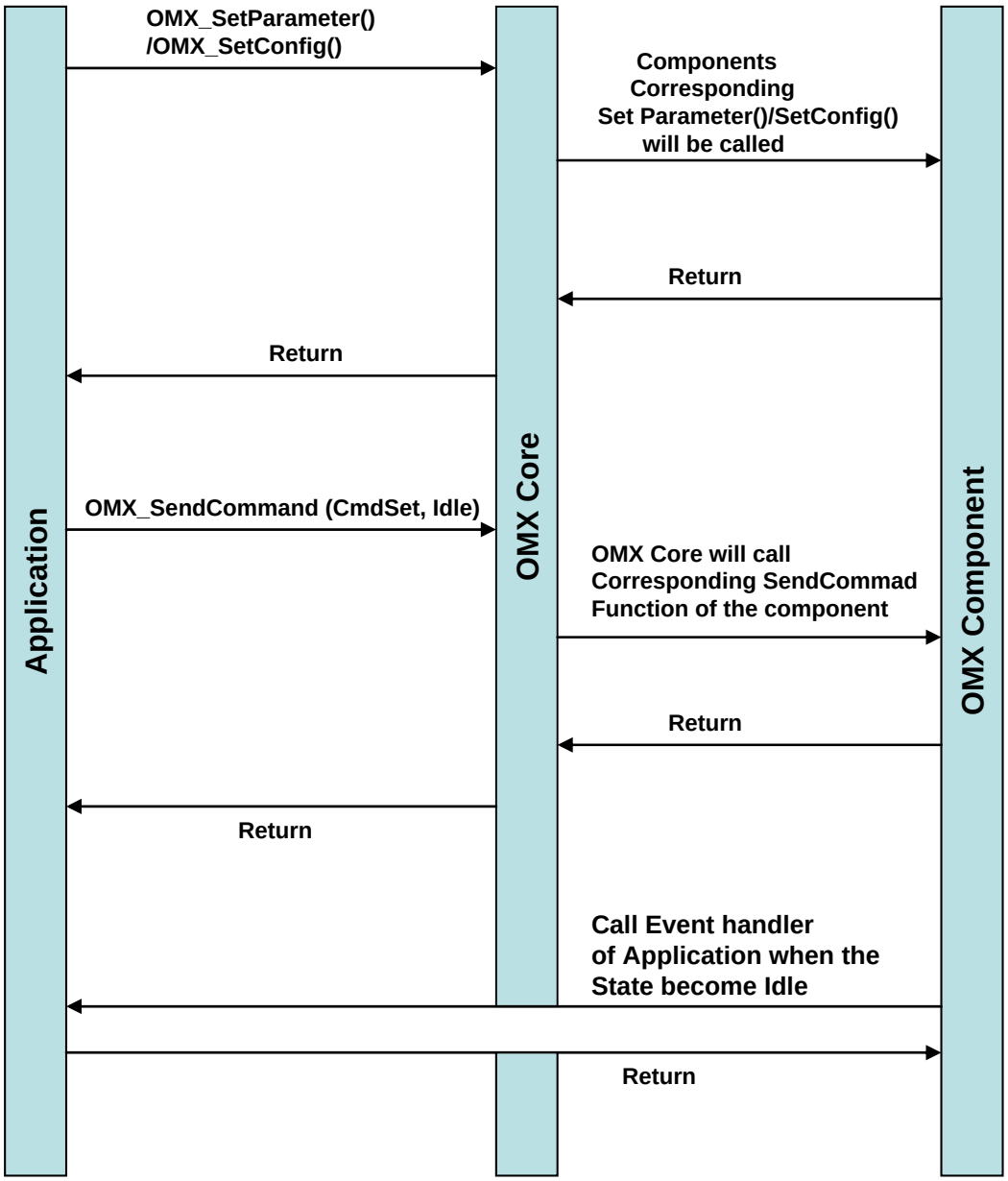


Figure 4 Initialization phase of Component

After all of the initialization structures are sent to the component via the OMX_SetConfig and OMX_SetParameter methods, the application will command the component to complete the initialization process. The component will then request any resources required for execution such as memory buffers, codec's, and component threads. In other words, the component is made ready for execution but the actual execution is not started. The Following sequences of actions describe how initialization is done.

- Prior to the initialization, the application will have obtained a valid component Handle.
- The application will allocate an initialization parameter structure and fill it with the desired values. Then the application will call the core's macro OMX_SetParameter and pass the filled initialization parameter structure as an argument to this macro. This macro in turn calls an appropriate function of the component which makes a local copy of this initialization parameter structure in the component's private data area. There are numerous kinds of settings or parameters that can be sent to the

component. Therefore, the second argument in function OMX_SetParameter specifies what kind of information the application intends to set in the component and the component will use these information/parameters later for initialization. This call is optional, as the application may chose to accept all of the default parameters by not sending any OMX_SetParameter structures.

- Similarly, the Application will call the core's macro OMX_SetConfig to configure the component with the desired settings. This call is also optional. OMX_SetConfig can be called any time after the component has been loaded.
- Next the Application sends idle command to the component using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateIdle. In response to this command, the component will allocate any remaining resources that are needed, will create the Pipes and threads needed for the component to function and will acquire and initialize the codec.
- The final action for Component Initialization is for the component to issue a callback to the application to notify the application that the initialization process has completed (with an error or successfully).

7.1.4 Component Execution

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.

Next the application will start the component by using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateExecuting. When the component reads this command, it will take appropriate action to start the codec. After the codec has been started, the component will issue all of the input buffers to the application and any output buffers to the codec. Actual processing will begin once the first input buffer has been received back from the application with input data. At the successful completion of this command, the component will send the OMX_HandleEvent callback to the application notifying the application that the component state has changed to executing state. Processing will continue until the application sends a buffer with last buffer flag (nFlags = `OMX_BUFFERFLAG_EOS`) or a command to put the component back to the initialized state (see component stop, section 7.1.7).

7.1.5 Component Pause

Prior to pausing the component, the application should have placed the component in the executing state. The application sends a pause command to the component using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StatePause. When the component's thread1 receives this command, it will take the appropriate action to pause the underlying codec and then stop sending buffers to the codec. It should be noted that if there are some buffers already being processed by the codec, they may be sent to the application. Once the processing of a buffer has started it is not paused mid-buffer. At the successful completion of this command, the component will send an OMX_HandleEvent callback to the application notifying the application that the component state has changed to the pause state.

7.1.6 Component Resume

The application must have placed the component into the pause state prior to issuing the resume command. The command OMX_StateExecuting acts as resume command when component is in pause state. The application sends resume command to component using core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateExecuting. When the component receives this command, it sends a corresponding command to the codec and resumes its execution and starts exchanging buffers with the application and codec again. At the successful completion of this command, the component will send an OMX_HandleEvent callback to the application notifying the application that the component state has changed to executing (resumed) state.

7.1.7 Component Stop

The entry criterion for this phase is that the component should be either in executing or paused state. The application sends the stop command to the component using the core's macro OMX_SendCommand with second argument set to OMX_CommandStateSet and third argument set to OMX_StateIdle. The application must continue to process buffers until the component responds with the OMX_HandleEvent callback indicating that it has transitioned to the Idle state. When the component receives this command, it sends a corresponding command to codec to stop the codec. The component then retrieves all of its buffers from the codec (and the application, if the application holds buffers). It is the application's responsibility to continue to process component buffers after sending the stop command so that all buffers get returned to the component. It should be noted that the component will not change state to idle state until all the buffers have been returned to the component. At the successful completion of this command, the component will send the OMX_HandleEvent callback to the application notifying the application that the component state has changed to the idle state.

7.1.8 Component Deinit

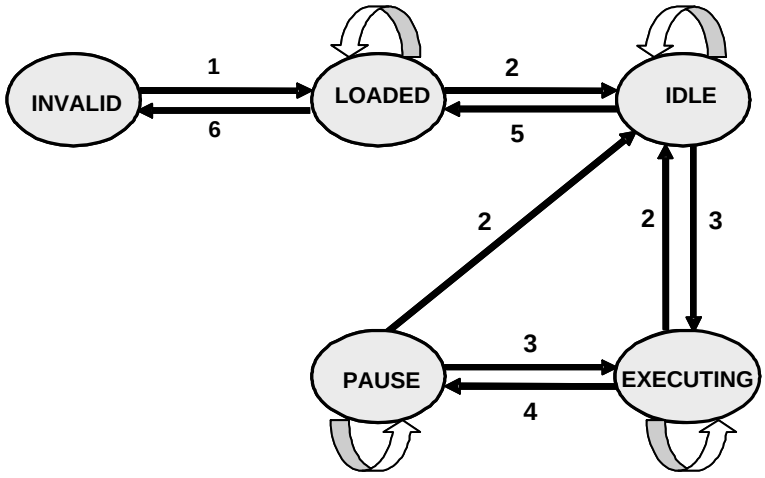
The entry criterion for this phase is that the component should be in the idle state. Once the application stops the component (by sending idle command) the component will go to the idle state. In order to de-initialize the component, the application should then call OMX_SendCommand with the second argument set to OMX_CommandStateSet and the third argument set to OMX_StateLoaded. This call will cause the component to try to release all of its resources. If the input parameters are invalid, the method will return with an error. In all other error cases, the component will try to complete the de-initialization task and to return without an error message. The only resources not released will be the component private data block. All codec resources, threads and data buffers will be released. Once these resources are released, the component will send the OMX_HandleEvent callback to the application notifying the application that the component state has changed to loaded state.

7.2 Component States

An OpenMAX™ TI 1.5 compliant component will exist in one of five states at any given time. Component states are controlled by the application via the OMX_SendCommand macro. Figure 5 represents the state diagram for OpenMAX™ TI 1.5 component. 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 the 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 application tries to load it using OMX_GetHandle. If component goes in the INVALID state due to corruption of its data structures etc then component can not change its state. In this case the component should be de-initialized and unloaded and then should be loaded again.

The states are defined in Table 5 of section 7.2.1.



1 OMX_GetHandle()
2 OMX_SendCommond (hComponent, OMX_CommandStateSet, OMX_StateIdle)
3 OMX_SendCommond (hComponent, OMX_CommandStateSet, OMX_StateExecuting)
4 OMX_SendCommond (hComponent, OMX_CommandStateSet, OMX_StatePause)
5 OMX_SendCommond (hComponent, OMX_CommandStateSet, OMX_StateLoaded)
6 OMX_FreeHandle()

Figure 5 State Diagram of a Component

7.2.1 State Definitions

The component shall be in one of the states defined in Table 5 at all times.

Table 5 OMX States

State	Description
LOADED State	This state specifies that the component has been loaded (initialized), a valid component handle has been given to the application, the application's callbacks have been registered with the component and memory has been allocated to the component's private data structures.
IDLE State	This state specifies that the component has been initialized which means that the component's threads have been created, buffers to be shared with codec have been allocated, and the codec has been initialized with application specified parameters.
EXECUTING State	This state specifies that the component is able to actively process the buffers and commands/messages. This means, buffers are being processed by component and they are being exchanged between the application, the component and the codec.
PAUSE State	This state specifies that the component is ready to perform processing, but has stopped processing buffers while waiting for a resume command. No further buffers will be processed by the component/codec except any buffers which were already being processed by codec when component was paused. Note that the pause command specifies that no new buffers will be processed.
INVALID State	This state specifies that the component's internal data structures have been corrupted and the component is not able to identify its state or that the component is in the process of being loaded/unloaded. If the component reaches this state unexpectedly, the component should be de-initialized and unloaded and then should be loaded again.
Wait for Resources state	This state is not used in OpenMAX™ TI 1.5, but is reserved for compatibility with the final Khronos OMX 1.0. Transitions to this state will not occur in OpenMAX™ TI 1.5

7.2.2 Valid State Transitions

OpenMAX components shall transition from one state to another state only in accordance with Table 6.

Table 6 OpenMAX™ TI 1.5 Component Transitions

Current State	Event/Command	Next State	Return value
OMX_STATE_INVALID (when component is not loaded/initialized)	Successful OMX_StateLoaded Command	OMX_STATE_LOADED	OMX_ErrorNone
OMX_STATE_INVALID (when component is not loaded/initialized)	Unsuccessful OMX_StateLoaded Command	OMX_STATE_INVALID	OMX_ErrorInvalidComponent/ OMX_ErrorInsufficientResources
OMX_STATE_INVALID (when component data structures have been corrupted due to some reason)	Any command/event	OMX_STATE_INVALID	OMX_ErrorUndefined.
OMX_STATE_LOADED	Successful OMX_StateIdle Command	OMX_STATE_IDLE	OMX_ErrorNone
OMX_STATE_LOADED	Unsuccessful OMX_StateIdle Command	OMX_STATE_LOADED	OMX_ErrorHardware/ OMX_ErrorInsufficientResources
OMX_STATE_LOADED	Other than OMX_StateIdle command	OMX_STATE_LOADED	OMX_ErrorInvalidState

Current State	Event/Command	Next State	Return value
OMX_STATE_IDLE	Successful OMX_StateExecuting Command	OMX_STATE_EXECUTING	OMX_ErrorNone
OMX_STATE_IDLE	Successful OMX_StateLoaded Command	OMX_STATE_LOADED	OMX_ErrorNone
OMX_STATE_IDLE	Other than OMX_StateExecuting/OMX_StateLoaded Command	OMX_STATE_IDLE	OMX_ErrorInvalidState
OMX_STATE_IDLE	Unsuccessful OMX_StateExecuting/OMX_StateLoaded Command	OMX_STATE_IDLE	OMX_ErrorHardware/ OMX_ErrorBadParameter
OMX_STATE_EXECUTING	Successful OMX_StatePause Command	OMX_STATE_PAUSE	OMX_ErrorNone
OMX_STATE_EXECUTING	Successful OMX_StateIdle Command	OMX_STATE_IDLE	OMX_ErrorNone
OMX_STATE_EXECUTING	Other than OMX_StatePause/OMX_StateIdle Command	OMX_STATE_EXECUTING	OMX_ErrorInvalidState
OMX_STATE_EXECUTING	Unsuccessful OMX_StatePause/OMX_StateIdle Command	OMX_STATE_EXECUTING	OMX_ErrorHardware
OMX_STATE_PAUSE	Successful OMX_StateExecuting Command	OMX_STATE_EXECUTING	OMX_ErrorNone
OMX_STATE_PAUSE	Successful OMX_StateIdle Command	OMX_STATE_IDLE	OMX_ErrorNone
OMX_STATE_PAUSE	Other than OMX_StateIdle/OMX_StateExecuting Command	OMX_STATE_PAUSE	OMX_ErrorInvalidState
OMX_STATE_PAUSE	Unsuccessful OMX_StateIdle/OMX_StateExecuing Command	OMX_STATE_PAUSE	OMX_ErrorHardware
Any State	Any error that causes component to not to be able to process data	OMX_STATE_INVALID	OMX_ErrorUndefined

8 Software Requirements

The requirements for the OpenMAX™ TI 1.5 Core are listed in Table 7.

Table 7 Requirements List

SR Tag	Requirement text	Application	Components	FB
SR14062	OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification			

9 Requirements Traceability

This section describes the Data types, data structures, callbacks and macros which are supported by the OpenMAX™ TI 1.5.

9.1 Class Structure

The OpenMAX™ TI 1.5 Core is written in C and not C++ and thus does not use classes. Refer section 9.3 for the data structures.

9.2 Defined Types

9.2.1 Basic Data Types

OMX core 1.5 exports set of data types which are given in Table 8. OMX core 1.5 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.

Table 8 OMX Basic Data Types

Type	Description
OMX_U8	Unsigned 8 bit byte, byte aligned
OMX_S8	Signed 8 bit byte, byte aligned
OMX_U16	Unsigned 16 bit word, word aligned
OMX_S16	Signed 16 bit word, word aligned
OMX_U32	Unsigned 32 bit double word, double word aligned
OMX_S32	Singed 32 bit double word, double word aligned
OMX_STRING	The OMX_STRING type is intended to be used to pass "C" type strings between the application and the core and component. The OMX_STRING type is a 32 bit pointer to a zero terminated string.
OMX_PTR	The OMX_PTR type is intended to be used to pass pointers between the OpenMAX™ TI 1.5 applications and the OpenMAX™ TI 1.5 Core and components.
OMX_BOOL	The OMX_BOOL type is intended to be used to represent TRUE or FALSE
OMX_BYTE	The OMX_BYTE type is intended to be used to pass arrays of bytes.
OMX_HANDLETYPE	Define the public interface for the OpenMAX™ TI 1.5 Handle. The core will not use this value internally, but the application should only use this value.

9.3 Data Structures

9.3.1 OMX_BUFFERHEADERTYPE

This is the structure that is passed for data transfer between the component and application. The details of the individual parameters are described in following table.

Table 9 OMX_BUFFERHEADERTYPE Structure

Data field Name	Description
OMX_U32 nSize	Size of the structure in bytes.
OMX_VERSIONTYPE nVersion	OpenMAX™ specification version information.
OMX_U8* pBuffer	Pointer to actual block of memory that is acting as the buffer.
OMX_U32 nAllocLen	Size of the buffer allocated, in bytes.
OMX_U32 nFilledLen	Number of bytes currently available in the buffer.
OMX_PTR pPortDefinition	It is not used in OpenMAX™ TI 1.5. The component will maintain the port definition information in its private data area. The port definition information is sent by application by calling SetParameter and in turn, component copies this information in its private data area.
OMX_PTR pComponentPrivate	Pointer to data that component wants to associate with this buffer. This is component's private data and can be accessed by the owner component only.
OMX_PTR pAppPrivate	Pointer to data that the application wants to associate with this buffer. This data is accessed by the application only.
OMX_PTR pBufferMark	This is not used in OpenMAX™ TI 1.5.
OMX_U32 nTickCount	This is not used in OpenMAX™ TI 1.5.
OMX_U32 nFlags	This flag indicates whether the buffer is the last buffer or not. At the time of buffer creation, the component sets its value to zero. This flag should be set to OMX_BUFFERFLAG_EOS to indicate the last buffer. The OMX_BUFFERFLAG_EOS is a macro that is defined in the OMX core header file OMX_Core.h (See the section 5.1.1).

9.3.2 OMX_COMPONENTTYPE

The OMX_COMPONENTTYPE structure defines the component handle. The component handle is used to access the component's public methods and also contains pointers to the component's private data area. The component handle is initialized by the OMX core (with the help of the component) during the process of loading/OMX_GetHandle of the component. After the component is successfully loaded, the application can safely access any of the component's public functions by using macros provided to the application. The application should not access the component methods using the component handle directly. This structure data will be valid until the component is unloaded by OMX_FreeHandle.

Table 10 OMX_COMPONENTTYPE Structure

Data field Name	Description or Evaluation
OMX_U32 nSize	The size of this structure, in bytes.

Data field Name	Description or Evaluation
OMX_VERSIONTYPE nVersion	The version of the OMX specification that the structure is built against.
OMX_PTR pComponentPrivate	The pointer to the component private data area.
OMX_PTR pApplicationPrivate	Unused by OMX Core. Value is set to NULL. The Application may use this field.
OMX_ERRORTYPE (*GetComponentVersion)	Function handler for the component method GetComponentVersion
OMX_ERRORTYPE (*SendCommand)	Function handler for the component method SendCommand
OMX_ERRORTYPE (*GetParameter)	Function handler for the component method GetParameter
OMX_ERRORTYPE (*SetParameter)	Function handler for the component method SetParameter
OMX_ERRORTYPE (*GetConfig)	Function handler for the component method GetConfig
OMX_ERRORTYPE (*SetConfig)	Function handler for the component method SetConfig
OMX_ERRORTYPE (*GetState)	Function handler for the component method GetState
OMX_ERRORTYPE (*ComponentTunnelRequest)	Function handler for the component method ComponentTunnelRequest
OMX_ERRORTYPE (*EmptyThisBuffer)	Function handler for the component method EmptyThisBuffer
OMX_ERRORTYPE (*FillThisBuffer)	Function handler for the component method FillThisBuffer
OMX_ERRORTYPE (*SetCallbacks)	Function handler for the component method SetCallbacks
OMX_ERRORTYPE (*ComponentDeInit)	Function handler for the component method ComponentDeInit. This is never called by the application. This method is only called by the OMX core.

9.3.3 OMX_BU32

This structure is used to hold the boundary limits for UNSIGNED data.

Table 11 OMX_BU32 Structure

Data Field Name	Description or Evaluation
OMX_U32 nValue	actual value
OMX_U32 nMin	minimum for value (i.e. nValue >= nMin)
OMX_U32 nMax	maximum for value (i.e. nValue <= nMax)

9.3.4 OMX_BS32

This structure is used to maintain the boundary limits for SIGNED data.

Table 12 OMX_BS32 Structure

Data Field Name	Description or Evaluation
OMX_S32 nValue	actual value
OMX_S32 nMin	minimum for value (i.e. nValue >= nMin)
OMX_S32 nMax	maximum for value (i.e. nValue <= nMax)

9.4 Unions

OpenMAX™ TI 1.5 core has one union data type.

9.4.1 OMX_VERSIONTYPE

The OMX_VERSIONTYPE union is used to specify the version of a structure or component. This data type is a member of all the OpenMAX™ TI 1.5 compliant data structures for the purpose of maintaining the version of each data structure. The version information should be filled by the entity that allocates the structure.

Table 13 OMX_VERSIONTYPE Union

Data field Name		Description or Evaluation
OMX_U32 nVersion	OMX_U8 nVersionMajor	Major version specifies an update in the functionality of the component or OpenMAX™ TI 1.5 specifications.
	OMX_U8 nVersionMinor	Minor version specifies an update in the component that does not change the functionality.
	OMX_U8 nRevision	Always 0 in the release version for TI standard.
	OMX_U8 nStep	Always 0 in the release version for TI standard.

9.5 Enumerations

This section describes all the enumeration data types declared in the OMX core. The OMX core design requires enumeration to support 32-bit enumeration data that is specified in OpenMAX™ TI 1.5 specifications. These data types are declared in OMX_Core.h and OMX_Types.h files.

9.5.1 OMX_COMMANDTYPE

This enumeration data type is used to specify the command passed by the application to the component.

Table 14 OMX_COMMANDTYPE Enumeration Data

Data field Name	Description or Evaluation
OMX_CommandStateSet	This command is passed to change the state of the component.
OMX_CommandMax	Do not use; needed by some compilers to force the enum size to be 32 bits.

9.5.2 OMX_STATETYPE

This enumeration data type is used for the following purpose,

- To specify the current state of the component.
- Used with OMX_SendCommand macro to send a state of transition to the component.

Table 15 OMX_STATETYPE Enumeration Data

Data field Name	Description or Evaluation
OMX_StateInvalid	This state indicates either the component is not loaded or component data structures are corrupted during any state of the component.
OMX_StateLoaded	This state indicates that component has been loaded but it is not initialized yet. In this state, only SetParameter/GetParameter and SetConfig/GetConfig methods of component can be called.
OMX_StateIdle	This state indicates that component has been initialized successfully and it is ready to start buffer processing.
OMX_StateExecuting	This state indicates that component is in execution state i.e. it is processing buffers and messages/commands.
OMX_StatePause	This state indicates that component has been paused and it is not processing any buffers.
OMX_StateMax	Do not use; needed by some compilers to force the enum size to be 32 bits.

9.5.3 OMX_ERRORTYPE

This enumeration enumerates all the error types that can be returned by OpenMAX™ TI 1.5 compliant components and the OpenMAX™ TI 1.5 core.

Table 16 OMX_ERRORTYPE Enumeration Data

Data field Name	Description or Evaluation
OMX_ErrorNone	Successful return
OMX_ErrorInsufficientResources	There were insufficient resources to perform the requested operation
OMX_ErrorUndefined	There was an error, but the cause of the error could not be determined
OMX_ErrorInvalidComponentName	The component name was not valid
OMX_ErrorComponentNotFound	No component with the specified name string was found. This error does not have scope in the static loading of components.
OMX_ErrorInvalidComponent	The component specified did not have “OMX_ComponentInit” or “OMX_ComponentDeInit entry point. This error does not have scope in the state of loading of components, as it is resolved in compile time itself.
OMX_ErrorBadParameter	One or more parameters were not valid
OMX_ErrorNotImplemented	The requested function is not implemented.
OMX_ErrorUnderflow	The buffer was emptied before the next buffer was ready.
OMX_ErrorOverflow	The buffer was not available when it was needed.

Data field Name	Description or Evaluation
OMX_ErrorHardware	The hardware failed to respond as expected.
OMX_ErrorTimeout	There was a timeout that occurred.
OMX_ErrorInvalidState	The component was in an invalid state for the command sent.
OMX_ErrorStreamCorrupt	Stream is found to be corrupt.
OMX_ErrorPortsNotCompatible	Ports being connected are not compatible.
OMX_ErrorResourcesLost	Resources allocated to an initialized component have been lost resulting in the component returning to the loaded state.
OMX_ErrorNoMore	This error is returned by component to indicate no more indices can be enumerated. This error message will be seen in cases where there is an array of structures (such as bands in an equalizer) and a band that is outside the valid range is selected.
OMX_ErrorVersionMismatch	Component detected a version mismatch.
OMX_ErrorNotReady	Component is not ready to return data at this time.
OMX_ErrorMax	Do not use; needed by some compilers to force the enum size to be 32 bits.

9.5.4 OMX_EVENTTYPE

This enumeration data type is used to represent an event that might occur in the component. This is used as one of the parameters when the component calls the application's callback EventHandle to notify the application that an event occurred.

Table 17 OMX_STATETYPE Enumeration Data

Data field Name	Description or Evaluation
OMX_EventStateChange	This specifies that the state of the component has changed.
OMX_EventError	This specifies that an error has occurred in performing some operation. An appropriate error number is returned to application through the callback.
OMX_EventMax	Do not use; needed by some compilers to force the enum size to be 32 bits.

9.5.5 OMX_BOOL

The OMX_BOOL type is used to represent a true or a false value when passing parameters to/from the OMX core/components. The OMX_BOOL is a 32 bit quantity and is aligned on a 32 bit word boundary.

Table 18 OMX_BOOL Type Enumeration Data

Data Field Name	Description or Evaluation
OMX_FALSE = 0	False enumeration.

Data Field Name	Description or Evaluation
OMX_TRUE = !OMX_FALSE	True enumeration.
OMX_BoolMax	Do not use; needed by some compilers to force the enum size to be 32 bits.

9.5.6 OMX_DIRTYPE

This is used to indicate if a port is an input port or output port. This enumeration is common across all component types.

Table 19 OMX_DIRTYPE Type Enumeration Data

Data Field Name	Description or Evaluation
OMX_DirInput	Port is an input port. The Buffer associated with OMX_DirInput specifies that buffer is input to the component from application.
OMX_DirOutput	Port is an output port. The Buffer associated with OMX_DirOutput specifies that buffer is output of the component to the application.
OMX_DirMax	Do not use; needed by some compilers to force the enum size to be 32 bits

9.5.7 OMX_ENDIANTYPE

The OMX_ENDIANTYPE enumeration is used to indicate the bit ordering for numerical data (i.e. big endian, or little endian)

Table 20 OMX_ENDIANTYPE Type Enumeration Data

Data Field Name	Description or Evaluation
OMX_EndianBig	Big endian.
OMX_EndianLittle	Little endian.
OMX_EndianMax	Do not use; needed by some compilers to force the enum size to be 32 bits

9.5.8 OMX_NUMERICALDATATYPE

The OMX_NUMERICALDATATYPE enumeration is used to indicate whether data is signed or unsigned.

Table 21 OMX_NUMERICALDATATYPE Type Enumeration Data

Data Field Name	Description or Evaluation
OMX_NumericalDataSigned	Signed data
OMX_NumericalDataUnsigned	Unsigned data
OMX_NumercialDataMax	Do not use; needed by some compilers to force the enum size to be 32 bits

9.6 API Requirements Coverage

9.6.1 OMX_GetHandle

```
OMX_ERRORTYPE OMX_GetHandle (OMX_HANDLETYPE* pHandle,
                             OMX_STRING pComponentName,
                             OMX_PTR pAppData,
                             OMX_CALLBACKTYPE* pCallBacks)
```

Implementation

The application calls this function to get the handle of the specified component. This function does following in sequence:

- Locate the specified component name in the OMX core maintained Component Table.
- If the component entry is found, allocates memory for the component handle which is of type OMX_COMPONENTTYPE.
- Call the registered function pointer OMX_ComponentInit of the component to get various function pointers of the component into the allocated component handle. This way, the component handle gets populated with various function pointers. The function OMX_ComponentInit also allocates memory for the component's private data structure and checks for the availability of any hardware if needed for component execution.
- Calls the function SetCallbacks of the component to send the application's callbacks to the component so that the component can use them when it is started.
- Returns the handle of the component if all of the above operations are successful else return an appropriate error to the application.

Parameters

Name	Type	Description
pHandle	OUT	This is the output argument which is filled by the component.
pComponentName	IN	This holds the name of the component that is to be loaded.
pAppData	IN	This specifies the application defined data. Since an application may drive multiple components, therefore, for each component, application defines a value which is unique to the component. This way whenever the application receives a callback from a component, the application checks the value of pAppData argument and gets to know which component this callback came from.
pCallBacks	IN	This is a pointer to a structure that holds the function pointers of all the application callbacks. These callbacks are registered with the component.

Return

OMX_ErrorNone	This is returned if macro executes successfully.
OMX_ErrorBadParameter	This error is returned if one of the input parameters is wrong.
OMX_ErrorInvalidComponentName	This error is returned if the component name exceeds the maximum length.
OMX_ErrorInsufficientResources	This error is returned if the core failed to allocate the memory for the component handle.
OMX_ErrorInvalidComponent	This error is returned if the core failed to find the component name in the component table list.

Pre Condition

OMX core has been initialized with OMX_Init.

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.6.2 OMX_FreeHandle

```
OMX_ERRORTYPE OMX_FreeHandle (OMX_HANDLETYPE hComponent)
```

Implementation

The application calls this function to free the component handle if it is no longer needed. This function does the following:

- Calls the function OMX_ComponentDeInit of the component to free the resource allocated to component's private data block. Note that except the component's private data block, all of the resources owned by the component are released in de-initialization phase of the component which is explained in section 4.1.8.
- Frees the memory allocated to the component handle.

After successfully returning from this function, the component handle is no longer valid to use.

Parameters

Name	Type	Description
hComponent	OUT	This specifies component handle that is to be freed by the OMX core.

Return

OMX_ErrorNone	This is returned if the macro executes successfully.
OMX_ErrorBadParameter	This error is returned if the input handle is not found in the array of handles.

Pre Condition

The component should be in loaded state.

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.6.3 OMX_SetupTunnel

```
OMX_ERRORTYPE OMX_SetupTunnel (OMX_HANDLETYPE hOutput,
                               OMX_U32 nPortOutput,
                               OMX_HANDLETYPE hInput,
                               OMX_U32 nPortInput)
```

Implementation

This function is not supported by the OpenMAX™ TI 1.5 and should return OMX_ErrorNotImplemented.

Parameters

Name	Type	Description
N/A		

Return

OMX_ErrorNotImplemented — This is returned since this function is not implemented in the OpenMAX™ TI 1.5 core.

Pre Condition

N/A

Post Condition

N/A

Requirement Coverage

- N/A

9.6.4 OMX_Init

```
OMX_ERRORTYPE OMX_Init ()
```

Implementation

The role of this function is to initialize the OMX core. Currently, this function is not required in the OpenMAX™ TI 1.5 and will return no error.

Parameters

Name	Type	Description
None		

Return

OMX_ErrorNone — Currently this function always returns this error number.

Pre Condition

This should be the first call to any OMX Core and should be called only once.

Post Condition

The OMX Core is ready for use.

Requirement Coverage

This method addresses requirement:

- N/A

9.6.5 OMX_Deinit

```
OMX_ERRORTYPE OMX_Deinit ()
```

Implementation

The role of this function is to de-initialize the OMX core. Currently, this function is not required in the OpenMAX™ TI 1.5 and will return no error.

Parameters

Name	Type	Description
None		

Return

OMX_ErrorNone	Currently this function always returns this error number.

Pre Condition

The application is ready to shutdown the OMX Core.

Post Condition

All OMX Resources have been released.

Requirement Coverage

This method addresses requirement:

- N/A

9.7 Macros

The OpenMAX™ TI 1.5 core provides a set of macros that are used by the application to perform various operations like loading the component, communicating with OMX component etc. These macros are defined in OMX_Core.h. Detailed description of each macro is given in following sub sections.

The common prerequisite for calling all the macros is to have a valid component handle. Note that following macro tables contain commonly returned error codes. If there are more error codes which need to be returned, these tables will be updated accordingly

9.7.1 OMX_GetComponentVersion

```
#define OMX_GetComponentVersion(        \
        hComponent,                     \
        pComponentName,                 \
        pComponentVersion,              \
        pSpecVersion,                   \
        pComponentUUID)
```

Implementation

This macro queries the component and returns information about the component.

Parameters

Name	Type	Description
hComponent	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.

Name	Type	Description
pComponentUUID	OUT	This is a pointer to the DSP_UUID structure which is be filled by the component.

Return

OMX_ErrorNone	This is returned if the macro executes successfully.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.

Pre Condition

The component handle should be valid to call this macro**.**

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.2 OMX_SendCommand

```
#define OMX_SendCommand(                    \
        hComponent,                         \
        Cmd,                                \
        Param)                              \
```

Implementation

This macro is used to send a command to the component. This is asynchronous call. The component checks all the input parameters and if they are valid, the command is queued to component's command pipe. When component gets chance to read command from the command pipe, the component will take the appropriate action based on the command and then it will call the application's HandleEvent callback.

Parameters

Name	Type	Description
hComponent	IN	This input argument is component handle.
Cmd	IN	This specifies the command type/category. The value of this argument can be OMX_CommandStateSet. Currently in OpenMAX™ TI 1.5, the only valid command is to set the component state.
Param	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

OMX_ErrorNone	This is returned if the macro executes successfully.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.
OMX_ErrorInvalidState	Indicates that the state specified by the argument Param is invalid for the current state of the component i.e. the component can not change to the state specified by the argument Param.

Pre Condition

If the value of the second argument (Cmd) is OMX_CommandStateSet, various preconditions for the state change are explained in section 5.

The application should have a valid component handle.

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.3 OMX_GetParameter

```
#define OMX_GetParameter(                       \
        hComponent,                             \
        nParamIndex,                            \
        ComponentParameterStructure)
```

Implementation

This macro gets the current parameter settings of the component. Since each type of component (audio, video) has different parameter settings and a particular component may have different kinds of settings, the OMX core defines various structures for all possible kinds of parameter settings. The application should allocate memory for the correct structure and pass it to the core using this macro.

Parameters

Name	Type	Description
hComponent	IN	This input argument is the component handle.
nParamIndex	IN	This identifies the structure being used by the third argument of the macro.
ComponentParameterStructure	OUT	This is a pointer to the structure which needs to be filled in by the component.

Return

OMX_ErrorNone	This output argument is returned when the component gives the required information.
OMX_ErrorBadParameter	This is returned if any arguments are invalid.

Pre Condition

The application should have a valid component handle.

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

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.4 OMX_SetParameter

```
#define OMX_SetParameter(                    \
        hComponent,                          \
        nParamIndex,                         \
        ComponentParameterStructure)
```

Implementation

This macro sets the various parameters of the component with desired values. Since each type of component (audio, video) has different parameter settings and a particular component may have different kinds of settings, the OMX core defines various structures for all possible kinds of parameter settings. The application should allocate memory for the correct structure, fill it with the required values and pass it to core using this macro. The component makes a local copy of this structure and uses it at the time of initialization. The OMX_SetParameter macro should be used to set the initialization parameters of the component, when the component is in the LOADED state.

Parameters

Name	Type	Description
hComponent	IN	This input argument is the component handle.
nParamIndex	IN	This identifies the structure being used by the third argument of the macro.
ComponentParameterStructure	IN	This input argument is a pointer to a structure which the component uses to make its local copy.

Return

OMX_ErrorNone	This is returned when component gives the required information.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.

Pre Condition

The application should have a valid component handle.

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

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.5 OMX_GetConfig

```
#define OMX_GetConfig (                      \
        hComponent,                          \
        nConfigIndex,                        \
        ComponentConfigStructure)
```

Implementation

This macro gets the configuration parameters of the component. This macro can be invoked at any time after the component has been loaded. Since each type of component (audio, video) has different configuration settings and a particular component may have different kinds of configuration settings, the OMX core defines various structures for all possible kinds of configuration settings. The application allocates the required structure and passes it to the core using this macro. The component fills this structure with the required information.

Parameters

Name	Type	Description
hComponent	IN	This input argument is the component handle.
nConfigIndex	IN	This identifies the structure being used by the third argument of the macro. The component checks the value of this argument to know which structure the application wants filled.
ComponentParameterStructure	OUT	This output argument is the pointer to the structure to be filled by the component.

Return

OMX_ErrorNone This is returned when the component gives the required information.

OMX_ErrorBadParameter This is returned when the one of the arguments is invalid.

Pre Condition

The application should have a valid component handle.

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

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.6 OMX_SetConfig

```
#define OMX_SetConfig (                                 \
        hComponent,                                     \
        nConfigIndex,                                   \
        ComponentConfigStructure)
```

Implementation

This macro sets the application specified configuration parameters of the component. This macro can be invoked any time after the component has been loaded. This is asynchronous call which means that a successful return from this call does not ensure that the component has been configured with the application specified configuration settings. Instead, it means that all validity checking has been done on the various parameters and the corresponding command has been written in the component's command pipe. As soon as the component reads this command, it will configure the codec with the application specified values.

Since each type of component (audio, video) has different configuration settings and a particular the component may have different kinds of configuration settings, the OMX core defines various structures for all possible kinds of configuration settings. The application allocates the required structure, fills it with desired values and passes it to the core using this macro. The component configures the codec with the application specified values.

If there are multiple calls to OMX_SetConfig, and the component has not yet configured the codec with previous call's configuration settings, the component will return an appropriate error code indicating that the component is busy configuring codec with previous settings. If the component is able to configure the codec successfully then the component will not call the application's event handle callback to notify application that codec has been configured. But if some error occurs while configuring the codec, the component does call event handler of the application to notify that an error has occurred while configuring the codec. This is done since chances of getting an error while configuring the codec is very minimal.

This OMX_SetConfig macro can be used at any time after the component has been loaded into the memory.

Parameters

Name	Type	Description
hComponent	IN	This input argument is the component handle.
nConfigIndex	IN	This identifies the structure being used by the third argument of the macro. The component checks the value of this argument to know which structure the application wants to be filled.
ComponentParameterStructure	IN	This input argument is a pointer to a structure that holds the values with which codec is to be configured

Return

OMX_ErrorNone	This is returned when the component gives the required information.
OMX_ErrorBadParameter	This is returned when one of the arguments is invalid.
OMX_ErrorInsufficientResources	This indicates that the codec is busy in performing previous configuration settings hence can not be configured this time.
OMX_ErrorTimeout	This indicates that a timeout has occurred while configuring the codec.

Pre Condition

The application should have a valid component handle.

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

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.7 OMX_GetState

```
#define OMX_GetState(                                   \
        hComponent,                                     \
        pState)
```

Implementation

The application calls this macro to get the current state of the component. This macro, in turn, invokes the corresponding function of the component to get the current state and stores the state in output argument pState.

Parameters

Name	Type	Description
hComponent	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

OMX_ErrorNone This is returned when the component gives the required information.

OMX_ErrorBadParameter This is returned when one of the arguments is invalid.

Pre Condition

The application should have a valid component handle.

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.8 OMX_EmptyThisBuffer

```
#define OMX_EmptyThisBuffer(                    \
        hComponent,                             \
        nPortIndex,                             \
        pBuffer)
```

Implementation

The application uses this macro to send a buffer filled with input data to the input port of the component. This is asynchronous call which means that the buffer will not be emptied instantly when component receives this call. Instead, the buffers is written in component's data pipe and later when component gets chance to read this buffer, it will empty this buffer and notify the application using the application's callback function EmptyBufferDone. In the component life cycle, there is always a callback EmptyBufferDone for each call to OMX_EmptyThisBuffer.

Note that terminology "EmptyThisBuffer" means the application is sending an input data buffer to the component for processing. The amount of data is specified by one of the element in the buffer header pointer specified by the third argument.

Parameters

Name	Type	Description
hComponent	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

OMX_ErrorNone This is returned when the component gives the required information.

OMX_ErrorPortsNotCompatible This is returned if the specified port index is not valid.

OMX_ErrorBadParameter This is returned when one of the arguments is invalid.

Pre Condition

The application should have a valid component handle.

The state of the component should be OMX_StateExecuting.

Post Condition

The application should always receive a callback EmptyBufferDone. Note that EmptyBufferDone is an element of the structure OMX_CALLBACKTYPE which contains all the callback function pointers of the application.

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.9 OMX_FillThisBuffer

```
#define OMX_FillThisBuffer(                                    \
        hComponent,                                             \
        nPortIndex,                                             \
        pBuffer)
```

Implementation

The Application uses this macro to send an empty buffer to the output port of the component. Before invoking this macro, the application must have received the buffer with the FillThisBufferDone callback from the component. This is asynchronous call which means that the buffer will not be filled with the output data instantly when the component receives this call. Instead, the buffer is written to the component's data pipe and later when the component gets a chance to read the data, it will fill this buffer and notify the application using the application's callback function FillBufferDone. In the component's life cycle, there is always a callback FillBufferDone for each call to OMX_FillThisBuffer except when component's buffers are returned to it at the time of stop command.

Note that the terminology "FillThisBuffer" means the application is sending an empty buffer to the component's output port to get the output data. The component will fill the buffer with output data and specify the amount of data in the buffer header.

Parameters

Name	Type	Description
hComponent	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

OMX_ErrorNone This is returned when the component gives the required information.

OMX_ErrorPortsNotCompatible This is returned if the specified port index is not valid.

OMX_ErrorBadParameter This is returned when one of the arguments is invalid

Pre Condition

The application should have a valid component handle.

The state of the component should be OMX_StateExecuting.

Post Condition

The application should receive a callback FillBufferDone except for the case when the application has issued a stop (i.e idle) command and trying to return the buffer to the component. In this case, component will not make a callback FillBufferDone to the application. Note that FillBufferDone is an element of the structure OMX_CALLBACKTYPE which contains all the callback function pointers of the application.

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.10 OMX_CALLBACKTYPE

This structure contains the callback function pointers of the application. The application allocates memory for this structure and sets its respective members with appropriate addresses. This structure is passed to the OMX core when the application calls the function OMX_GetHandle so that the OMX core can provide the same information to the component which will finally need the application's callbacks for buffer exchange. After returning from OMX_GetHandle, the application can free the memory allocated to this structure since the application no longer needs to maintain it.

9.7.10.1 EventHandler

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

Implementation

The EventHandler method is used to notify the application when an event of interest occurs. This event may be change of state, an error occurred etc. The application will get the event notification for the events which are listed in the section 9.5.4.

Parameters

Name	Type	Description
hComponent	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

Pre Condition

None

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.10.2 EmptyBufferDone

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

Implementation

This is the callback function of the application that a component uses to return an empty input buffer for the application for use. There is always a callback EmptyBufferDone from the component for each OMX_EmptyThisBuffer call from the application except for one case. This case is when the component is started since at this time, the application has no access to buffers hence they have to come from nowhere but the component.

Parameters

Name	Type	Description
hComponent	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

Pre Condition

None

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.7.10.3 FillBufferDone

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

Implementation

This is the application callback function that the component uses to return a filled output buffer to application. There is always a callback FillBufferDone from the component for each call OMX_FillThisBuffer from the application.

Parameters

Name	Type	Description
hComponent	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

None

Pre Condition

None

Post Condition

None

Requirement Coverage

This method addresses requirement:

SR14062: OMX core shall comply with Texas Instruments OpenMAX IL V1.5 specification.

9.8 Non-API Requirements Coverage

This OpenMAX™ TI 1.5 core will comply with the TI coding guidelines, located in the Clear Case® VOB path:

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