JPEG Encoder on OMAP3530

User's Guide

Literature Number: SPRUFN8
December 2008

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

Products		**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
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

Preface

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) JPEG Encoder implementation on the OMAP3530 platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI's codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

Intended Audience

This document is intended for system engineers who want to integrate TI's codecs with other software to build a multimedia system based on the OMAP3530 platform.

This document assumes that you are fluent in the C language, have a good working knowledge of Digital Signal Processing (DSP), digital signal processors, and DSP applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM) standard will be helpful.

How to Use This Manual

This document includes the following chapters:

- **Chapter 1 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the codec and lists its supported features.
- **Chapter 2 - Installation Overview**, describes how to install, build, and run the codec.
- **Chapter 3 - Sample Usage**, describes the sample usage of the codec.
- **Chapter 4 - API Reference**, describes the data structures and interface functions used in the codec.
- **Appendix A – Standard Huffman Table**, provides an example for initialization of a structure - type `IDMJPGE_TIGEM_CustomHuffmanTables` using the Standard Huffman Table.

- **Appendix B – Standard Quantization Table**, provides an example for initialization of a structure type `IDMJPGE_TIGEM_CustomQuantTables` using the Standard Quantization Table.

Related Documentation From Texas Instruments

The following documents describe TI's DSP algorithm standards such as, XDAIS and XDM. To obtain a copy of any of these TI documents, visit the Texas Instruments website at www.ti.com.

- *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interface Standard (also known as XDAIS) specification.
- *Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (literature number SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.
- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577) describes how an eXpressDSP-compliant algorithm may be used effectively in a static system with limited memory.
- *DMA Guide for eXpressDSP-Compliant Algorithm Producers and Consumers* (literature number SPRA445) describes the DMA architecture specified by the TMS320 DSP Algorithm Standard (XDAIS). It also describes two sets of APIs used for accessing DMA resources: the IDMA2 abstract interface and the ACPY2 library.
- *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8)

The following documents describe TMS320 devices and related support tools:

- *Design and Implementation of an eXpressDSP-Compliant DMA Manager for C6X1X* (literature number SPRA789) describes a C6x1x-optimized (C6211, C6711) ACPY2 library implementation and DMA Resource Manager.
- *TMS320C64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.
- *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871) describes the C64x+ megamodule peripherals.
- *TMS320C64x to TMS320C64x+ CPU Migration Guide* (literature number SPRAA84) describes migration from the Texas Instruments

TMS320C64x™ digital signal processor (DSP) to the TMS320C64x+™ DSP.

- *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (literature number SPRU187N) explains how to use compiler tools such as compiler, assembly optimizer, standalone simulator, library-build utility, and C++ name demangler.
- *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.

Related Documentation

You can use the following documents to supplement this user guide:

- *ISO/IEC IS 10918-1 Information Technology - Digital Compression and Coding of Continuous-Tone Still Images -- Part 1: Requirements and Guidelines* | *CCITT Recommendation T.81*

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
COFF	Common Object File Format
CSL	Chip Support Library
DCT	Discrete Cosine Transform
DHT	Define Huffman Table
DQT	Define Quantization Table
DRI	Define Restart Interval
DSP	Digital Signal Processing
EVM	Evaluation Module
IJG	Independent JPEG Group
JFIF	JPEG File Interchange Format
JPEG	Joint Photographic Experts Group
MCU	Minimum Coded Unit
QDMA	Quick Direct Memory Access
SOF	Start of Frame

Abbreviation	Description
SOI	Start of Image
SOS	Start of Scan
VLD	Variable Length Decoding
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media

Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (‘‘) represents pseudo-code.
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

Product Support

When contacting TI for support on this codec, quote the product name (JPEG Baseline Profile Encoder on OMAP3530) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Code Composer Studio, DSP/BIOS, eXpressDSP, TMS320, TMS320C64x, TMS320C6000, TMS320DM644x, and TMS320C64x+ are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the JPEG Encoder on the OMAP3530 platform and its supported features.

Topic **Page**

1.1 Overview of XDAIS and XDM

TI's multimedia codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

1.1.1 XDAIS Overview

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. In order to facilitate these functionalities, the IALG interface defines the following APIs:

- `algAlloc()`
- `algInit()`
- `algActivate()`
- `algDeactivate()`
- `algFree()`

The `algAlloc()` API allows the algorithm to communicate its memory requirements to the client application. The `algInit()` API allows the algorithm to initialize the memory allocated by the client application. The `algFree()` API allows the algorithm to communicate the memory to be freed when an instance is no longer required.

Once an algorithm instance object is created, it can be used to process data in real-time. The `algActivate()` API provides a notification to the algorithm instance that one or more algorithm processing methods is about to be run zero or more times in succession. After the processing methods have been run, the client application calls the `algDeactivate()` API prior to reusing any of the instance's scratch memory.

The IALG interface also defines three more optional APIs `algControl()`, `algNumAlloc()`, and `algMoved()`. For more details on these APIs, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

1.1.2 XDM Overview

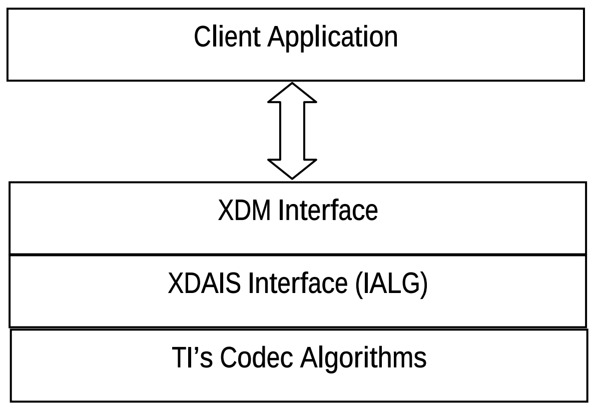
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the client application, it is important that all codecs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs (for example audio, video, image, and speech). The XDM standard defines the following two APIs:

- `control()`
- `process()`

The `control()` API provides a standard way to control an algorithm instance and receive status information from the algorithm in real-time. The `control()` API replaces the `algControl()` API defined as part of the IALG interface. The `process()` API does the basic processing (encode/decode) of data.

Apart from defining standardized APIs for multimedia codecs, XDM also standardizes the generic parameters that the client application must pass to these APIs. The client application can define additional implementation specific parameters using extended data structures.

The following figure depicts the XDM interface to the client application.



As depicted in the figure, XDM is an extension to XDAIS and forms an interface between the client application and the codec component. XDM insulates the client application from component-level changes. Since TI's multimedia algorithms are XDM-compliant, it provides you with the flexibility to use any TI algorithm without changing the client application code. For example, if you have developed a client application using an XDM-compliant MPEG4 video decoder, then you can easily replace MPEG4 with another XDM-compliant video decoder, say H.263, with minimal changes to the client application.

For more details, see *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8).

1.2 Overview of JPEG Encoder

The JPEG Encoder accepts planar image data in YUV4:2:0, YUV4:1:1, YUV4:2:2, and YUV4:4:4 formats. It accepts interleaved image data in YUV4:2:2 format and accepts grayscale input.

1.3 Supported Services and Features

This user guide accompanies TI's implementation of JPEG Encoder on the OMAP3530 platform.

This version of the codec has the following supported features:

- Supports baseline sequential mode for interleaved data formats (single scan)
- Supports multiple scans for planar formats YUV420, YUV411, YUV422, and YUV444
- Supports arbitrary image size
- Supports a maximum of three scans
- Supports comment insertion into the JPEG header
- Supports frame-based mode encoding
- Includes a standard JPEG header and also supports JFIF style header
- Supports custom Huffman and Quantization tables
- Supports DRI marker insertions in the compressed bit-stream
- Quantization tables are fixed with a quality factor (1-100) adjusting the quantization level
- eXpressDSP Digital Media (XDM 1.0 IIMGENC1) compliant
- Does not support encoding of images with pixel resolution more than 8 bits per pixel
- Supports thumbnail
- Supports insertion of application data APP0, APP1, and APP13
- This codec uses the iVLC hardware accelerator for implementations

Chapter 2

Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing the codec component. It also provides information on building and running the sample test application.

2.1 System Requirements

This section describes the hardware and software requirements for the normal functioning of the codec component.

2.1.1 Hardware

This codec has been built and tested on the OMAP3530 EVM with XDS560 JTAG emulator.

2.1.2 Software

The following are the software requirements for the normal functioning of the codec:

- **Development Environment:** This project is developed using Code Composer Studio 3.3.49.
- **Code Generation Tools:** This project is compiled, assembled, archived, and linked using the code generation tools version 6.0.14.

2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 100_I_JPEG_E_2_01, under which another directory named OMAP3530_BL_001 is created.

Figure 2-1 shows the sub-directories created in the OMAP3530_BL_001 directory.

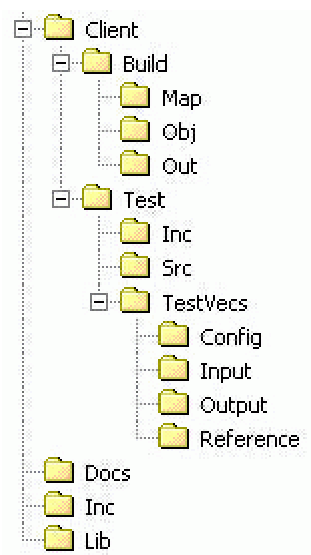


Figure 2-1. Component Directory Structure

> **Note:**
>
> If you are installing an evaluation version of this codec, the directory name will be 100E_I_JPEG_E_2_01

Table 2-1 provides a description of the sub-directories created in the OMAP3530_BL_001 directory.

Table 2-1. Component Directories

Sub-Directory	Description
\Inc	Contains XDM related header files which allow interface to the codec library
\Lib	Contains the codec library file
\Docs	Contains user guide and datasheet
\Client\Build	Contains the sample test application project (.pjt) file
\Client\Build\Map	Contains the memory map generated on compilation of the code
\Client\Build\Obj	Contains the intermediate .asm and/or .obj file generated on compilation of the code
\Client\Build\Out	Contains the final application executable (.out) file generated by the sample test application
\Client\Test\Src	Contains application C files
\Client\Test\Inc	Contains header files needed for the application code
\Client\Test\TestVecs\Input	Contains input test vectors
\Client\Test\TestVecs\Output	Contains output generated by the codec
\Client\Test\TestVecs\Reference	Contains read-only reference output to be used for verifying codec output
\Client\Test\TestVecs\Config	Contains configuration parameter files

2.3 Before Building the Sample Test Application

This codec is accompanied by a sample test application. To run the sample test application, you need DSP/BIOS and TI Framework Components (FC).

This version of the codec has been validated with DSP/BIOS version 5.31.02 and Framework Component (FC) version 2.20.01.

2.3.1 Installing DSP/BIOS

You can download DSP/BIOS from the TI external website:

https://www-a.ti.com/downloads/sds_support/targetcontent/bios/index.html

Install DSP/BIOS at the same location where you have installed Code Composer Studio. For example:

<install directory>\CCStudio_v3.3

The sample test application uses the following DSP/BIOS files:

- Header file, bcache.h available in the <install directory>\CCStudio_v3.3\<bios_directory>\packages \ti\bios\include directory.

- Library file, biosDM420.a64P available in the <install directory>\CCStudio_v3.3\<bios_directory>\packages \ti\bios\lib directory.

2.3.2 Installing Framework Component (FC)

You can download FC from the TI external website:

https://www-a.ti.com/downloads/sds_support/targetcontent/FC/index.html

Extract the FC zip file to the same location where you have installed Code Composer Studio. For example:

<install directory>\CCStudio_v3.3

The test application and the library use the following header files:

- ialg.h available in the <install directory>\CCStudio_v3.3\framework_components_2_20_01\fctools\packages\ti\xdais\ directory

- iimgenc1.h available in the <install directory>\CCStudio_v3.3\framework_components_2_20_01\fctools\packages\ti\xdais\dm\ directory

- xdas.h available in the <install directory>\CCStudio_v3.3\framework_components_2_20_01\fctools\packages\ti\xdais\ directory

- xdm.h available in the <install directory>\CCStudio_v3.3\framework_components_2_20_01\fctools\packages\ti\xdais\dm\ directory
- idma3.h available in the directory>\CCStudio_v3.3\framework_components_2_20_01\fctools\packages\ti\xdais\
- dman3.h available in the directory>\CCStudio_v3.3\framework_components_2_20_01\packages\ti\sdo\fc\dman3\

2.4 Building and Running the Sample Test Application

The sample test application that accompanies this codec component will run in TI's Code Composer Studio development environment. To build and run the sample test application in Code Composer Studio, follow these steps:

1) Verify that you have installed of TI's Code Composer Studio version 3.3.49 and code generation tools version 6.0.14.

2) Verify that the codec object library dmjpge_tigem.l64P exists in the \Lib sub-directory.

3) Open the test application project file, TestAppEncoder.pjt in Code Composer Studio. This file is available in the \Client\Build sub-directory.

4) Select the configuration as **FILE_IO**.

5) Select **Project > Build** to build the sample test application. This creates an executable file, TestAppEncoder.out in the \Client\Build\Out sub-directory.

6) Select **File > Load**, browse to the \Client\Build\Out sub-directory, select the codec executable created in step 4, and load it into Code Composer Studio in preparation for execution.

7) Select **Debug > Run** to execute the sample test application.

   The sample test application takes the input files stored in the \Client\Test\TestVecs\Input sub-directory, runs the codec, and uses the reference files stored in the \Client\Test\TestVecs\Reference sub-directory to verify that the codec is functioning as expected.

8) On successful completion, the application displays one of the following messages for each frame:

   o "Encoder compliance test passed/failed" (for compliance check mode)

   o "Encoder output dump completed" (for output dump mode)

2.5 Configuration Files

This codec is shipped along with:

- Generic configuration file (Testvecs.cfg) - specifies input and reference files for the sample test application.
- Encoder configuration file (Testparams.cfg) - specifies the configuration parameters used by the test application to configure the Encoder.

2.5.1 Generic Configuration File

The sample test application shipped along with the codec uses the configuration file, Testvecs.cfg for determining the input and reference files for running the codec and checking for compliance. The Testvecs.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

The format of the Testvecs.cfg file is:

```
X
Config
Input
Output/Reference
```

where:

- `X` may be set as:
  - 1 - for compliance checking, no output file is created
  - 0 - for writing the output to the output file
- `Config` is the Encoder configuration file. For details, see Section 2.5.2.
- `Input` is the input file name (use complete path).
- `Output/Reference` is the output file name (if `X` is 0) or reference file name (if `X` is 1).

A sample Testvecs.cfg file is as shown:

```
1
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\Input_422.yuv
..\..\Test\TestVecs\Reference\Output_422.jpg
0
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\Input_422.yuv
..\..\Test\TestVecs\Output\Output_422.jpg
```

2.5.2 Encoder Configuration File

The encoder configuration file, Testparams.cfg contains the configuration parameters required for the encoder. The Testparams.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

A sample Testparams.cfg file is as shown:

```
# Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
###########################################################
Parameters
###########################################################
MaxImageWidth = 768        # Image width in Pels
MaxImageHeight = 512       # Image height in Pels
IPImageWidth = 768         # Image width in Pels
IPImageHeight = 512        # Image height in Pels
Scan = 0                   # 0 =>default
ForceChromaFormat = 2      # 0 =>default
                           # 1 => XDM_YUV_420P,
                           # 2 => XMI_YUV_422P,
                           # 3 => XDM_YUV_422IBE,
                           # 4 => XDM_YUV_422ILE ,
                           # 5 => XMI_YUV_444P ,
                           # 6 => XMI_YUV_411P
                           # 7 => XDM_GRAY ,
                           # 8 => XDM_RGB
inputChromaFormat = 4      # 1 => XDM_YUV_420P,
                           # 2 => XMI_YUV_422P ,
                           # 3 => XDM_YUV_422IBE,
                           # 4 => XDM_YUV_422ILE ,
                           # 5 => XMI_YUV_444P ,
                           # 6 => XMI_YUV_411P
                           # 7 => XDM_GRAY ,
                           # 8 => XDM_RGB
QP = 73                    # 0 => Min, 100=> Max
numMCU = 0                 # 0 for Frame based encoding else
                               slice based encoding
captureWidth = 0           # Capture width in Pels, must be
                             equal to 0 or equal to
                             Inputwidth
captureHeight = 0          # Capture Height  must be equalto
                             0 or equal to actual
                             Inputheight
DRI_Interval = 0           # 0 - DRI Marker is not inserted
                             other than 0 - will add DRI
                             marker at regular interval
```

Any field in the `IIMGENC1_Params` structure (see Section 4.2.1.5) can be set in the Testparams.cfg file using the syntax shown above. If you specify additional fields in the Testparams.cfg file, ensure to modify the test application appropriately to handle these fields.

2.6 Standards Conformance and User-Defined Inputs

To check the conformance of the codec for the default input file shipped along with the codec, follow the steps as described in Section 2.4.

To check the conformance of the codec for other input files of your choice, follow these steps:

1) Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory.

2) Copy the reference files to the \Client\Test\TestVecs\Reference sub-directory.

3) Edit the configuration file, Testvecs.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the Testvecs.cfg file, see Section 2.5.1.

4) Execute the sample test application. On successful completion, the application displays one of the following messages for each frame:

   o "Encoder compliance test passed/failed" (if `X` is 1)

   o "Encoder output dump completed" (if `X` is 0)

If you have chosen the option to write to an output file (`X` is 0), you can use any standard file comparison utility to compare the codec output with the reference output and check for conformance.

2.7 Uninstalling the Component

To uninstall the component, delete the codec directory from your hard disk.

2.8 Evaluation Version

If you are using an evaluation version of this codec, a Texas Instruments logo will be visible in the output.

Chapter 3

Sample Usage

This chapter provides a detailed description of the sample test application that accompanies this codec component.

3.1 Overview of the Test Application

The test application exercises the `IIMGENC1` base class of the JPEG Encoder library. The main test application files are TestAppEncoder.c and TestAppEncoder.h. These files are available in the \Client\Test\Src and \Client\Test\Inc sub-directories respectively.

Figure 3-1 depicts the sequence of APIs exercised in the sample test application.

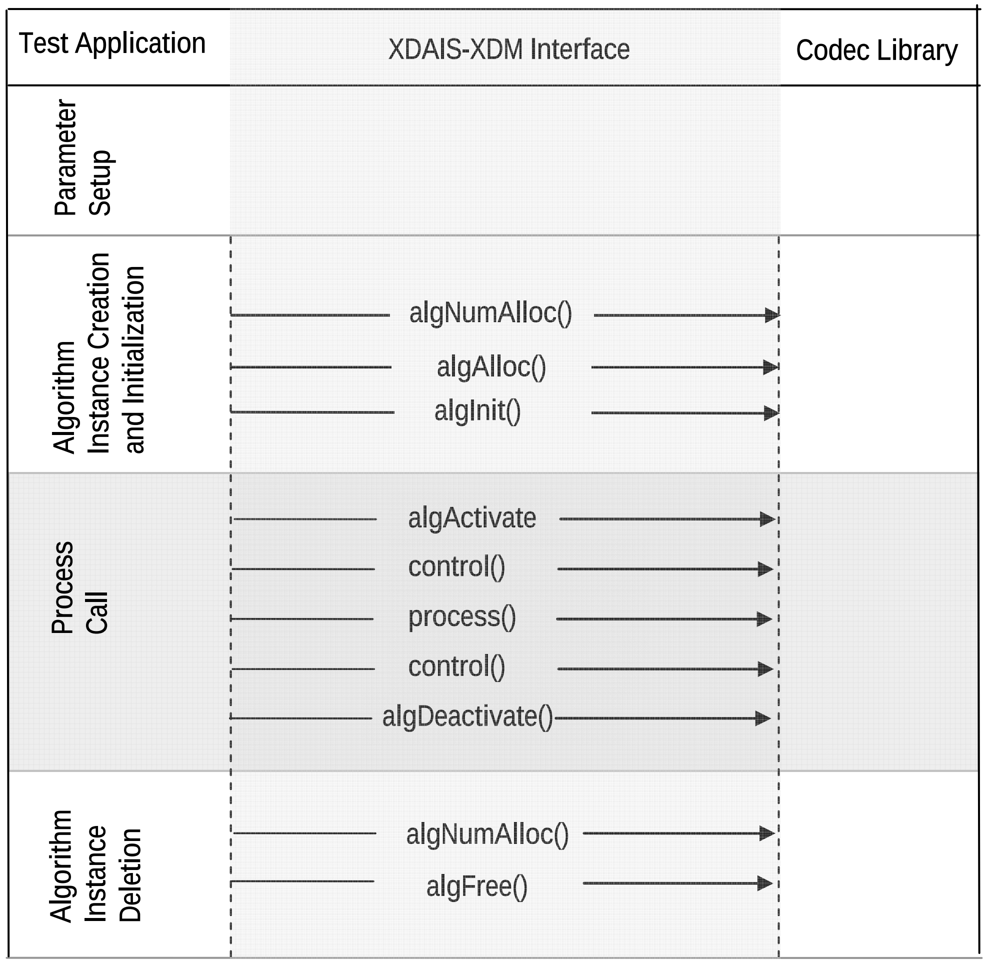


Figure 3-1. Test Application Sample Implementation

The test application is divided into four logical blocks:

- Parameter setup
- Algorithm instance creation and initialization
- Process call
- Algorithm instance deletion

3.1.1 Parameter Setup

Each codec component requires various codec configuration parameters to be set at initialization. For example, a video codec requires parameters such as video height, video width, and so on. The test application obtains the required parameters from the Encoder configuration files.

In this logical block, the test application performs the following:

1) Opens the generic configuration file, Testvecs.cfg and reads the compliance checking parameter, Encoder configuration file name (Testparams.cfg), input file name, and output/reference file name.

2) Opens the Encoder configuration file, (Testparams.cfg) and reads the various configuration parameters required for the algorithm.

   For more details on the configuration files, see Section 2.5.

3) Sets the `IIMGENC1_Params` structure based on the values it reads from the Testparams.cfg file.

4) Reads the input bit-stream into the application input buffer.

After successful completion of the above steps, the test application performs the algorithm instance creation and initialization.

3.1.2 Algorithm Instance Creation and Initialization

In this logical block, the test application accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs are called in sequence:

1) `algNumAlloc()` - To query the algorithm about the number of memory records it requires.

2) `algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.

3) `algInit()` - To initialize the algorithm with the memory structures provided by the application.

A sample implementation of the create function that calls `algNumAlloc()`, `algAlloc()`, and `algInit()` in sequence is provided in the `ALG_create()` function implemented in the alg_create.c file.

3.1.3 Process Call

After algorithm instance creation and initialization, the test application performs the following:

1) Sets the dynamic parameters (if they change during run-time) by calling the `control()` function with the `XDM_SETPARAMS` command.

2) Sets the input and output buffer descriptors required for the `process()` function call. The input and output buffer descriptors are obtained by calling the `control()` function with the `XDM_GETBUFINFO` command.

3) Calls the `process()` function to encode/decode a single frame of data. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.2.1.6). The inputs to the process function are input and output buffer descriptors, pointer to the `IIMGENC1_InArgs` and `IIMGENC1_OutArgs` structures.

The `control()` and `process()` functions should be called only within the scope of the `algActivate()` and `algDeactivate()` XDAIS functions which activate and deactivate the algorithm instance respectively. Once an algorithm is activated, there could be any ordering of `control()` and `process()` functions. The following APIs are called in sequence:

1) `algActivate()` - To activate the algorithm instance.
2) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.
3) `process()` - To call the Encoder with appropriate input/output buffer and arguments information.
4) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.
5) `algDeactivate()` - To deactivate the algorithm instance.

The do-while loop encapsulates frame level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts. It also protects the `process()` call from file operations by placing appropriate calls for cache operations as well. The test application does a cache invalidate for the valid input buffers before `process()` and a cache write back invalidate for output buffers after `process()`.

In the sample test application, after calling `algDeactivate()`, the output data is either dumped to a file or compared with a reference file.

3.1.4 Algorithm Instance Deletion

Once encoding/decoding is complete, the test application must delete the current algorithm instance. The following APIs are called in sequence:

1) `algNumAlloc()` - To query the algorithm about the number of memory records it used.
2) `algFree()` - To query the algorithm to get the memory record information and then free them up for the application.

A sample implementation of the delete function that calls `algNumAlloc()` and `algFree()` in sequence is provided in the `ALG_delete()` function implemented in the alg_create.c file.

Chapter 4

API Reference

This chapter provides a detailed description of the data structures and interfaces functions used in the codec component.

4.1 Symbolic Constants and Enumerated Data Types

This section summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. For each symbolic constant, the semantics or interpretation of the same is also provided.

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
XDM_DataFormat	XDM_BYTE	Big endian stream.
	XDM_LE_16	16-bit little endian stream.
	XDM_LE_32	32-bit little endian stream.
XDM_ChromaFormat	XDM_YUV_420P	YUV 4:2:0 planar
	XDM_YUV_422P	YUV 4:2:2 planar
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian). Not supported in this version of JPEG Encoder.
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian)
	XDM_YUV_444P	YUV 4:4:4 planar
	XDM_YUV_411P	YUV 4:1:1 planar
	XDM_GRAY	Gray format
	XDM_RGB	RGB color format. Not supported in this version of JPEG Encoder.
	XDM_DEFAULT	Default value
XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill the Status structure
	XDM_SETPARAMS	Set run-time dynamic parameters via the DynamicParams structure
	XDM_RESET	Reset the algorithm
	XDM_SETDEFAULT	Initialize all fields in Params structure to default values specified in the library
	XDM_FLUSH	Handle end of stream conditions. This command forces algorithm instance to output data without additional input.
	XDM_GETBUFINFO	Query algorithm instance regarding the properties of input and output buffers

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_SETIMGPREPROCESSOR	Image pipe is not supported in this version of JPEG Encoder.
	XDM_INSERTCOMMENT	Pass a BYTE array to insert into the JPEG header as a comment.
	XDM_GETVERSION	Retrieve a pointer to a CHAR array containing a string with the current library version.
XDM_EncMode	XDM_ENCODE_AU	Encode entire access unit
	XDM_GENERATE_HEADER	Encode only header
XDM_ErrorBit	XDM_APPLIEDCONCEALMENT	Bit 9 ❑ 1 - Applied concealment ❑ 0 - Ignore Not applicable for JPEG Encoder.
	XDM_INSUFFICIENTDATA	Bit 10 ❑ 1 - Insufficient data ❑ 0 - Ignore
	XDM_CORRUPTEDDATA	Bit 11 ❑ 1 - Data problem/corruption ❑ 0 - Ignore
	XDM_CORRUPTEDHEADER	Bit 12 ❑ 1 - Header problem/corruption ❑ 0 - Ignore Not applicable for JPEG Encoder.
	XDM_UNSUPPORTEDINPUT	Bit 13 ❑ 1 - Unsupported feature/parameter in input ❑ 0 - Ignore Not applicable for JPEG Encoder.
	XDM_UNSUPPORTEDPARAM	Bit 14 ❑ 1 - Unsupported input parameter or configuration ❑ 0 - Ignore This bit is set by the JPEG Encoder when an input parameter is not valid. For example, this can occur when the input data format is not supported.
	XDM_FATALERROR	Bit 15 ❑ 1 - Fatal error (stop encoding) ❑ 0 - Recoverable error This bit is set when a fatal error occurs. The JPEG Encoder cannot be used for an encode process in this state.

Note:

The remaining bits that are not mentioned in XDM_ErrorBit are interpreted as:

- Bit 16-32 : Reserved
- Bit 8 : Reserved
- Bit 0-7 : Codec and implementation specific (see Table 4-2)

The algorithm can set multiple bits to 1 depending on the error condition.

- Comment insertion is supported through Application buffer but not through XDM_INSERTCOMMENT control command
- XDM_GETVERSION is not supported in this version of JPEG Encoder. However, version can be extracted through vers.exe

The JPEG Encoder specific error status messages are listed in Table 4-2. The value column indicates the decimal value of the last 8-bits reserved for codec specific error statuses.

Table 4-2. JPEG Encoder Error Statuses

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
IJPEGENC_ErrorStatus	JPEGENC_SUCCESS	0	Successful encoding.
	JPEGENC_OUTPUT_BUFF_SIZE	1	Set an error, if output buffer size is less than required.
	JPEGENC_THUMB_RESOLUTION	2	Set an error, if Thumnail_resolutions is greater than Image_resolutions.

4.2 Data Structures

This section describes the XDM defined data structures that are common across codec classes. These XDM data structures can be extended to define any implementation specific parameters for a codec component.

4.2.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- XDM_BufDesc
- XDM1_BufDesc
- XDM_AlgBufInfo
- IIMGENC1_Fxns
- IIMGENC1_Params
- IIMGENC1_DynamicParams
- IIMGENC1_InArgs
- IIMGENC1_Status
- IIMGENC1_OutArgs

4.2.1.1 XDM_BufDesc

|| Description

This structure defines the buffer descriptor for input and output buffers.

|| Fields

Field	Data type	Input/ Output	Description
**bufs	XDAS_Int8	Input	Pointer to the vector containing buffer addresses
numBufs	XDAS_Int32	Input	Number of buffers
*bufSizes	XDAS_Int32	Input	Size of each buffer in bytes

4.2.1.2 XDM1_BufDesc

|| Description

This structure defines the single buffer descriptor for input and output buffers.

|| Fields

Field	Data type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
descs[XDM_MAX_IO_BUFFERS]	XDM1_SingleBufDesc	Input	Array of Buffer descriptors

4.2.1.3 XDM_AlgBufInfo

|| Description

This structure defines the buffer information descriptor for input and output buffers. This structure is filled when you invoke the `control()` function with the `XDM_GETBUFINFO` command.

|| Fields

Field	Data type	Input/ Output	Description
minNumInBufs	XDAS_Int32	Output	Number of input buffers
minNumOutBufs	XDAS_Int32	Output	Number of output buffers
minInBufSize[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Output	Size in bytes required for each input buffer

Field	Data type	Input/ Output	Description
minOutBufSize[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Output	Size in bytes required for each output buffer

Note:

The JPEG Encoder has the following buffer information:

- control() function must be called with XDM_CmdId set to XDM_SETPARAMS before calling with XDM_CmdId set to XDM_GETBUFINFO.
- Number of input buffers required are 1 for XDM_YUV_422ILE and 3 for planar formats.
- Number of output buffer required is 1.
- Set maxWidth/inputWidth and maxHeight/inputHeight as multiples of the number specified in the following table based on the input chromaformat.

Input Format	Multiplication Factor	
	Width	Height
XDM_YUV_420P	16	16
XDM_YUV_422P	16	8
XDM_YUV_422ILE	16	8
XDM_YUV_444P	8	8
XDM_YUV_411P	32	8
XDM_GRAY	8	8

- By default, each input buffer size is equal to (maxHeight * maxWidth * 2) for XDM_YUV_422ILE and (maxHeight * maxWidth *3) for planar formats. See IIMGENC1_Params data structure for details.
- Output buffer size can be estimated as Height * Width * Quality Factor /100
- For interleaved input format, output will have only one scan irrespective of the output format
- If the output format is set to XDM_GRAY, output will have only one scan irrespective of the input format.

4.2.1.4 IIMGENC1_Fxns

|| Description

This structure contains pointers to all the XDAIS and XDM interface functions.

|| Fields

Field	Data type	Input/ Output	Description
ialg	IALG_Fxns	Input	Structure containing pointers to all the XDAIS interface functions. For more details, see *TMS320 DSP Algorithm Standard API Reference* (SPRU360).
*process	XDAS_Int32	Input	Pointer to the `process()` function.
*control	XDAS_Int32	Input	Pointer to the `control()` function.

4.2.1.5 IIMGENC1_Params

|| Description

This structure defines the creation parameters for algorithm instance object. Set this data structure to `NULL`, if you are not sure of the values to be specified for these parameters.

|| Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes. The default value is `sizeof (IDMJPGE_TIGEM_Params)`
maxHeight	XDAS_Int32	Input	Maximum image height. The default value is 2000.
maxWidth	XDAS_Int32	Input	Maximum image width. The default value is 2000.
maxScans	XDAS_Int32	Input	Maximum number of scans. `maxScans` should be set to `XDM_DEFAULT`.
dataEndianness	XDAS_Int32	Input	Endianness of input data. The JPEG Encoder implementation supports only `XDM_BYTE` format.

Field	Data type	Input/ Output	Description
forceChromaFormat	XDAS_Int32	Input	Output color format for the encoder. See XDM_ChromaFormat enumeration for details. If set to XDM_DEFAULT, the output color format matches the input color format. Planar input formats can be converted to XDM_GRAY by ignoring the color planes. The default value is XDM_YUV_422P.

Note:

If the input is non-interleaved, then the output is non-interleaved. If the input is interleaved, then the output also is interleaved.

The output formats supported for each input format are as follows:

Input Format	Supported Output Formats
XDM_YUV_411P	❑ XDM_YUV_411P ❑ XDM_GRAY
XDM_YUV_420P	❑ XDM_YUV_420P ❑ XDM_GRAY
XDM_YUV_422P	❑ XDM_YUV422P ❑ XDM_GRAY
XDM_YUV_422ILE	❑ XDM_YUV_422P ❑ XDM_YUV_420P ❑ XDM_GRAY
XDM_YUV_444P	❑ XDM_YUV_444P ❑ XDM_GRAY

4.2.1.6 IIMGENC1_DynamicParams

|| Description

This structure defines the run-time parameters for algorithm instance object. Set this data structure to NULL, if you are not sure of the values to be specified for these parameters.

|| Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes. Default value: size of (IDMJPGE_TIGEM_DynamicParams)

Field	Data type	Input/ Output	Description
numAU	XDAS_Int32	Input	Number of access units to encode. Setting this field to XDM_DEFAULT encodes the complete frame. Any other value will encode the specified number of access units. Default value: XDM_DEFAULT
inputChromaFormat	XDAS_Int32	Input	Color format of the input data. See XDM_ChromaFormat enumeration for details. Default value: XDM_YUV_422P
inputHeight	XDAS_Int32	Input	Input height of the image. Default value: 480
inputWidth	XDAS_Int32	Input	Input width of the image Default value: 720
captureWidth	XDAS_Int32	Input	Total buffer width of the input data. This must be equal to or greater than inputWidth. Default value: 0
generateHeader	XDAS_Int32	Input	❑ XDM_ENCODE_AU - Encode entire access unit ❑ XDM_GENERATE_HEADER - Encode only header Default value: XDM_ENCODE_AU
qValue	XDAS_Int32	Input	Q value for encoder. Set to a value between 0 and 100 to define the quantization parameter. ❑ 1 - lowest quality ❑ 100 - highest quality Default value: 86

Note:

- To support smaller output buffer, dump the output after every process call, else should increment the output buffer by bytes generated after every process call. To support slice based encoding, set the NumAu to the required value.
- For efficient encoding, that is good quality and comparatively good compression, can be achieved by having the quality factor (qValue) in the range of 20 to 80.
- Set the appropriate output buffer size to insert application markers.
- To encode the portion of an image, set the inputWidth and inputHeight to the expected value. Set the captureWidth and captureHeight to the actual width and height of the image. Setting the captureWidth and captureHeight other than the actual value, will result in visual artifacts.

4.2.1.7 IIMGENC1_InArgs

|| Description

This structure defines the run-time input arguments for algorithm instance object.

|| Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.

4.2.1.8 IIMGENC1_Status

|| Description

This structure defines parameters that describe the status of the algorithm instance object.

|| Fields

Field	Datatype	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error enumeration. See XDM_ErrorBit enumeration for details.
totalAU	XDAS_Int32	Output	Total number of iterations for the encoder to encode the current image frame.
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information. See XDM_AlgBufInfo data structure for details.

4.2.1.9 IIMGENC1_OutArgs

|| Description

This structure defines the run-time output arguments for the algorithm instance object.

|| Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error enumeration. See XDM_ErrorBit enumeration for details.
bytesGenerated	XDAS_Int32	Output	Number of bytes generated in the current bit-stream.
currentAU	XDAS_Int32	Output	The current MCU number that the encoder has completed. This can be used to track the encoder status in the case of slice-based encoding.

4.2.2 JPEG Encoder Data Structures

This section includes the following JPEG Encoder specific extended data structures:

- IDMJPGE_TIGEM_Params
- IDMJPGE_TIGEM_DynamicParams
- IDMJPGE_TIGEM_InArgs
- IDMJPGE_TIGEM_Status
- IDMJPGE_TIGEM_OutArgs

4.2.2.1 IDMJPGE_TIGEM_Params

|| Description

This structure defines the creation parameters and any other implementation specific parameters for the JPEG Encoder instance object. The creation parameters are defined in the XDM data structure, IIMGENC1_Params.

|| Fields

Field	Data type	Input/ Output	Description
params	IIMGENC1_Params	Input	See IIMGENC1_Params data structure for details.
MaxThumbnail_H_size_App0	XDAS_UInt16	Input	Maximum horizontal resolution for APP0 thumbnail
MaxThumbnail_V_size_App0	XDAS_UInt16	Input	Maximum vertical resolution for APP0 thumbnail
MaxThumbnail_H_size_App1	XDAS_UInt16	Input	Maximum horizontal resolution for APP1 thumbnail
MaxThumbnail_V_size_App1	XDAS_UInt16	Input	Maximum vertical resolution for APP1 thumbnail
MaxThumbnail_H_size_App13	XDAS_UInt16	Input	Maximum horizontal resolution for APP13 thumbnail
MaxThumbnail_V_size_App13	XDAS_UInt16	Input	Maximum vertical resolution for APP13 thumbnail

4.2.2.2 IDMJPGE_TIGEM_DynamicParams

|| Description

This structure defines the run-time parameters and any other implementation specific parameters for a JPEG Encoder instance object. The run-time parameters are defined in the XDM data structure, `IIMGENC1_DynamicParams`.

|| Fields

Field	Datatype	Input/ Output	Description
`params`	`IIMGENC1_DynamicParams`	Input	See `IIMGENC1_DynamicParams` data structure for details.
`captureHeight`	`XDAS_Int32`	Input	If set to 0 , input height is equal to capture height , else capture height should be set to actual image height.
`quantTable`	`IDMJPGE_TIGEM_CustomQuantTables *`	Input	To support custom Quantization Table
`DRI_Interval`	`XDAS_Int32`	Input	❑ 0 - DRI Marker is not inserted ❑ Other than 0 -Insert DRI marker at regular interval in the compressed bit-stream.
`huffmanTable`	`IDMJPGE_TIGEM_CustomHuffmanTables*`	Input	To support custom huffman table
`APPN0_numBufs`	`XDAS_UInt8`	Input	Number of APP0 buffers
`APPN1_numBufs`	`XDAS_UInt8`	Input	Number of APP1 buffers
`APPN13_numbufs`	`XDAS_UInt8`	Input	Number of APP13 buffers
`APPN0_startBuf`	`XDAS_UInt8`	Input	Start buffer number in inbufs for APP0 buffers
`APPN1_startBuf`	`XDAS_UInt8`	Input	Start buffer number in inbufs for APP1 buffers
`APPN13_startBuf`	`XDAS_UInt8`	Input	Start buffer number in `inbufs` for APP13 buffers

Field	Datatype	Input/ Output	Description
COMMENT_startBuf	XDAS_UInt8	Input	Start buffer number in inbufs for Comment buffer
COMMENT_insert	XDAS_UInt8	Input	Specifies whether to insert comment or not
Thumbnail_Index_App0	XDAS_UInt8	Input	❑ 0 - Thumbnail not added in APP0 data ❑ 1 - Thumbnail appended in 1st APP0 data ❑ 2 - Thumbnail appended in 2nd APP0 data The maximum value is 16.
Thumbnail_Index_App1	XDAS_UInt8	Input	❑ 0 - Thumbnail not added in APP1 data ❑ 1 - Thumbnail appended in 1st APP1 data ❑ 2 - Thumbnail appended in 2nd APP1 data The maximum value is 16.
Thumbnail_Index_App13	XDAS_UInt8	Input	❑ 0- Thumbnail not added in APP13 data ❑ 1 - Thumbnail appended in 1st APP13 data ❑ 2 - Thumbnail appended in 2nd APP13 data The maximum value is 16.
Thumbnail_H_size_App0	XDAS_UInt16	Input	Horizontal resolution for APP0 thumbnail
Thumbnail_V_size_App0	XDAS_UInt16	Input	Vertical resolution for APP0 thumbnail
Thumbnail_H_size_App1	XDAS_UInt16	Input	Horizontal resolution for APP1 thumbnail
Thumbnail_V_size_App1	XDAS_UInt16	Input	Vertical resolution for APP1 thumbnail
Thumbnail_H_size_App13	XDAS_UInt16	Input	Horizontal resolution for APP13 thumbnail
Thumbnail_V_size_App13	XDAS_UInt16	Input	Vertical resolution for APP13 thumbnail

Note:

- Input Width, Input Height, Thumbnail Width, and Thumbnail Height should be in multiples of the value mentioned in the following table to have the sufficient data to encode. If not, data present in the padded buffer will be taken to encode and bit-exactness will not be assured for the same, as the data present in the padded buffer will not be the same always.

Input Format	Multiplication Factor	
	Width	Height
XDM_YUV_420P	16	16
XDM_YUV_422P	16	8
XDM_YUV_422ILE	16	8
XDM_YUV_444P	8	8
XDM_YUV_411P	32	8
XDM_GRAY	8	8

- Thumbnail resolution should be less then Image resolution.
- Minimum resolution of thumbnail is 16x16.
- Odd thumnail resolutions is not supported.

4.2.2.3 IDMJPGE_TIGEM_CustomHuffmanTables

|| Description

This structure defines the table format for custom Huffman table support and has two parts

- The first part used in the VLC Module defines the custom VLC tables that should be passed to the JPEG Encoder.
- The second part defines the format in which the custom Huffman tables should be passed to the JPEG Encoder compliant with the RFC 2035 (RTP Payload Format for JPEG-compressed Video) format. This second part creates the DHT marker in the JPEG Header.

This structure should be aligned to 128 byte boundary that is, the last 7 bits in the allocated base address should be zeroes. The `IDMJPGE_TIGEM_DynamicParams` structure contains a pointer to this structure of type `IDMJPGE_TIGEM_CustomHuffmanTables`. This field passes custom Huffman tables to the JPEG Encoder. The application just needs to initialize the field appropriately.

However, if the application requires to use the Standard Huffman Tables, then there are 2 ways of intimating this to the JPEG Encoder -

- Using only the base class `IIMGENC1_DynamicParams` and not the extended class `IDMJPGE_TIGEM_DynamicParams`, the `size` field in the DynamicParams structure should be set to the size of IIMGENC1_DynamicParams.

- Using the extended class IDMJPGE_TIGEM_DynamicParams, by setting the field IDMJPGE_TIGEM_CustomHuffmanTables *huffmanTable to NULL.

|| Fields

Field	Data type	Input/ Output	Description
lum_dc_vlc[12]	XDAS_Int32	Input	The array lum_dc_vlc defines VLC table for the luma DC component.
lum_ac_vlc[16][16]	XDAS_UInt32	Input	The array lum_ac_vlc defines VLC table for the luma AC component.
chm_dc_vlc[12]	XDAS_UInt32	Input	The array chm_dc_vlc defines VLC table for the chroma DC component.
chm_ac_vlc[16][16]	XDAS_UInt32	Input	The array chm_ac_vlc defines VLC table for the chroma AC component.
lum_dc_codelens[16]	XDAS_UInt8	Input	The array lum_dc_codelens defines the number of codewords corresponding to a coded symbol of given code length.
lum_dc_ncodes	XDAS_UInt16	Input	The variable lum_dc_ncodes defines the size of lum_dc_codelens, that is, 16
lum_dc_symbols[12]	XDAS_UInt8	Input	The array lum_dc_symbols defines the list of coded symbols.
lum_dc_nsymbols	XDAS_UInt16	Input	The variable lum_dc_nsymbols defines the number of coded symbols or actual size of lum_dc_symbols - maximum 12.
lum_ac_codelens[16]	XDAS_UInt8	Input	The array lum_ac_codelens defines the number of codewords corresponding to a coded symbol of given code length.
lum_ac_ncodes	XDAS_UInt16	Input	The variable lum_ac_ncodes defines the size of lum_ac_codelens, that is, 16.
lum_ac_symbols[162]	XDAS_UInt8	Input	The array lum_ac_symbols defines the list of coded symbols.
lum_ac_nsymbols	XDAS_UInt16	Input	The variable lum_ac_nsymbols defines the number of coded symbols or actual size of lum_ac_symbols - maximum 162.
chm_dc_codelens[16]	XDAS_UInt8	Input	The array chm_dc_codelens defines the number of codewords corresponding to a coded symbol of given code length.
chm_dc_ncodes	XDAS_UInt16	Input	The variable chm_dc_ncodes defines the size of chm_dc_codelens, that is, 16.

Field	Data type	Input/ Output	Description
chm_dc_symbols[12]	XDAS_UInt8	Input	The array chm_dc_symbols defines the list of coded symbols.
chm_dc_nsymbols	XDAS_UInt16	Input	The variable chm_dc_nsymbols defines the number of coded symbols or actual size of chm_dc_symbols - maximum 12.
chm_ac_codelens[16]	XDAS_UInt8	Input	The array chm_ac_codelens defines the number of codewords corresponding to a coded symbol of given code length.
chm_ac_ncodes	XDAS_UInt16	Input	The variable chm_ac_ncodes defines the size of chm_ac_codelens, that is, 16.
chm_ac_symbols[162]	XDAS_UInt8	Input	The array chm_ac_symbols defines the list of coded symbols.
chm_ac_nsymbols	XDAS_UInt16	Input	The variable chm_ac_nsymbols defines the number of coded symbols or actual size of chm_ac_symbols - maximum 162.

The Standard Huffman Table is given in Appendix A as an example initialization of the above.

Note:

- For lum_dc_vlc and chm_dc_vlc, the index is the size in bits of the DC coefficient. This can be from 0 to 11 (0x0 to 0xB). In each 32-bit word, the upper 16 bits define the length of the codeword and the lower 16 bits define the actual codeword.
- For lum_ac_vlc and chm_ac_vlc, it is a 2D array where the column index is the run-length of zeros that precede the non-zero quantized DCT coefficient. This can be from 0 to 15 (0x0 to 0xF). The row index is the size in bits of the non-zero coefficient that followed the run of zeros. This can be from 1 to 10 (0x1 to 0xA). This implies rows 1 to 10 will comprise of 16 X 10 = 160 codewords. Row 0 contains 2 special codewords - 0x00 and 0xF0. So, there are in total 162 possible AC code words. The standard huffman AC tables include all 162. Rows 11 to 15 should contains all zeroes. In each 32-bit word, the upper 16 bits define the length of the codeword and the lower 16 bits define the actual codeword.
- 0x00 represents an End of Block (EOB), which indicates that there are no more non-zero AC coefficients in this component, and that the encoder will move to the next component.
- 0xF0 represents a Zero Run Length (ZRL), which indicates that there was a run of > 15 zeros. This codeword represents a run of 15 zeros, and will be followed by another codeword that indicates another ZRL or a normal run + size codeword.

4.2.2.4 IDMJPGE_TIGEM_CustomQuantTables

|| Description

This structure defines the format for custom Quantization tables for both Luma and Chroma that needs to be passed to the JPEG Encoder and will be used in the Quantization Module. The format is as specified in RFC 2035 (RTP Payload Format for JPEG-compressed Video) and in Table K.1 and K.2 of JPEG Spec.

The `IDMJPGE_TIGEM_DynamicParams` structure contains a pointer to the structure type `IDMJPGE_TIGEM_CustomQuantTables`. This field is used to pass custom Quantization tables to the JPEG Encoder. The application has to initialize this field appropriately. However, if the application needs to use the Standard Quantization Tables, there are two ways of implementing this in JPEG Encoder -

- Using only the base class `IIMGENC1_DynamicParams` and not the extended class `IDMJPGE_TIGEM_DynamicParams`, in which case the `size` field in the `DynamicParams` structure should be set to the size of `IIMGENC1_DynamicParams`.
- Using the extended class `IDMJPGE_TIGEM_DynamicParams`, but by setting the field `IDMJPGE_TIGEM_CustomQuantTables *quant`Table to `NULL`.

|| Fields

Field	Data type	Input/ Output	Description
lum_quant_tab[64]	XDAS_UInt16	Input	The array lum_quant_tab defines the quantization table for the luma component.
chm_quant_tab[64]	XDAS_UInt16	Input	The array chm_quant_tab defines the quantization table for the chroma component.

The Standard Quantization Table is provided in Appendix B as an example initialization of the above.

4.2.2.5 IDMJPGE_TIGEM_InArgs

|| Description

This structure defines the run-time input arguments for JPEG Encoder instance object.

|| Fields

Field	Data type	Input/ Output	Description
inArgs	IIMGENC1_InArgs	Input	See IIMGENC1_InArgs data structure for details.

4.2.2.6 IDMJPGE_TIGEM_Status

|| Description

This structure defines parameters that describe the status of the JPEG Encoder and any other implementation specific parameters. The status parameters are defined in the XDM data structure, IIMGENC1_Status.

|| Fields

Field	Data type	Input/ Output	Description
status	IIMGENC1_Status	Output	See IIMGENC1_Status data structure for details.

4.2.2.7 IDMJPGE_TIGEM_OutArgs

|| Description

This structure defines the run-time output arguments for the JPEG Encoder instance object.

|| Fields

Field	Data type	Input/ Output	Description
outArgs	IIMGENC1_OutArgs	Output	See IIMGENC1_OutArgs data structure for details.

4.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the JPEG Encoder. The APIs are logically grouped into the following categories:

- **Creation** – `algNumAlloc()`, `algAlloc()`
- **Initialization** – `algInit()`
- **Control** – `control()`
- **Data processing** – `algActivate()`, `process()`, `algDeactivate()`
- **Termination** – `algFree()`

You must call these APIs in the following sequence:

1) `algNumAlloc()`
2) `algAlloc()`
3) `algInit()`
4) `algActivate()`
5) `process()`
6) `algDeactivate()`
7) `algFree()`

`control()` can be called any time after calling the `algInit()` API.

`algNumAlloc()`, `algAlloc()`, `algInit()`, `algActivate()`, `algDeactivate()`, and `algFree()` are standard XDAIS APIs. This document includes only a brief description for the standard XDAIS APIs. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

4.3.1 Creation APIs

Creation APIs are used to create an instance of the component. The term creation could mean allocating system resources, typically memory.

‖ Name

`algNumAlloc()` – determine the number of buffers that an algorithm requires

‖ Synopsis

```
XDAS_Int32 algNumAlloc(Void);
```

‖ Arguments

```
Void
```

‖ Return Value

```
XDAS_Int32; /* number of buffers required */
```

‖ Description

`algNumAlloc()` returns the number of buffers that the `algAlloc()` method requires. This operation allows you to allocate sufficient space to call the `algAlloc()` method.

`algNumAlloc()` may be called at any time and can be called repeatedly without any side effects. It always returns the same result. The `algNumAlloc()` API is optional.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

`algAlloc()`

‖ Name

algAlloc() – determine the attributes of all buffers that an algorithm requires

‖ Synopsis

```
XDAS_Int32 algAlloc(const IALG_Params *params, IALG_Fxns
**parentFxns, IALG_MemRec memTab[]);
```

‖ Arguments

```
IALG_Params *params; /* algorithm specific attributes */

IALG_Fxns **parentFxns;/* output parent algorithm
functions */

IALG_MemRec memTab[]; /* output array of memory records */
```

‖ Return Value

```
XDAS_Int32 /* number of buffers required */
```

‖ Description

algAlloc() returns a table of memory records that describe the size, alignment, type, and memory space of all buffers required by an algorithm. If successful, this function returns a positive non-zero value indicating the number of records initialized.

The first argument to algAlloc() is a pointer to a structure that defines the creation parameters. This pointer may be NULL; however, in this case, algAlloc() must assume default creation parameters and must not fail.

The second argument to algAlloc() is an output parameter. algAlloc() may return a pointer to its parent's IALG functions. If an algorithm does not require a parent object to be created, this pointer must be set to NULL.

The third argument is a pointer to a memory space of size nbufs * sizeof(IALG_MemRec) where, nbufs is the number of buffers returned by algNumAlloc() and IALG_MemRec is the buffer-descriptor structure defined in ialg.h.

After calling this function, memTab[] is filled up with the memory requirements of an algorithm.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

algNumAlloc(), algFree()

4.3.2 Initialization API

Initialization API is used to initialize an instance of the algorithm. The initialization parameters are defined in the Params structure (see Data Structures section for details).

‖ Name

`algInit()` – initialize an algorithm instance

‖ Synopsis

```
XDAS_Int32 algInit(IALG_Handle handle, IALG_MemRec
memTab[], IALG_Handle parent, IALG_Params *params);
```

‖ Arguments

```
IALG_Handle handle; /* algorithm instance handle*/

IALG_memRec memTab[]; /* array of allocated buffers */

IALG_Handle parent; /* handle to the parent instance */

IALG_Params *params; /* algorithm initialization
parameters */
```

‖ Return Value

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

‖ Description

`algInit()` performs all initialization necessary to complete the run-time creation of an algorithm instance object. After a successful return from `algInit()`, the instance object is ready to be used to process data.

The first argument to `algInit()` is a handle to an algorithm instance. This value is initialized to the base field of `memTab[0]`.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers allocated for an algorithm instance. The number of initialized records is identical to the number returned by a prior call to `algAlloc()`.

The third argument is a handle to the parent instance object. If there is no parent object, this parameter must be set to `NULL`.

The last argument is a pointer to a structure that defines the algorithm initialization parameters.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

‖ See Also

`algAlloc()`, `algMoved()`

4.3.3 Control API

Control API is used for controlling the functioning of the algorithm instance during run-time. This is done by changing the status of the controllable parameters of the algorithm during run-time. These controllable parameters are defined in the `Status` data structure (see Data Structures section for details).

|| Name

`control()` – change run-time parameters and query the status

|| Synopsis

```
XDAS_Int32 (*control) (IIMGENC1_Handle handle,
IIMGENC1_Cmd id, IIMGENC1_DynamicParams *params,
IIMGENC1_Status *status);
```

|| Arguments

`IIMGENC1_Handle handle; /* algorithm instance handle */`

`IIMGENC1_Cmd id; /* algorithm specific control commands*/`

`IIMGENC1_DynamicParams *params` /* `algorithm` run-time parameters */

`IIMGENC1_Status *status` /* `algorithm` instance status parameters */

|| Return Value

`IALG_EOK; /* status indicating success */`

`IALG_EFAIL; /* status indicating failure */`

|| Description

This function changes the run-time parameters of an algorithm instance and queries the algorithm's status. `control()` must only be called after a successful call to `algInit()` and must never be called after a call to `algFree()`.

The first argument to `control()` is a handle to an algorithm instance.

The second argument is an algorithm specific control command. See `XDM_CmdId` enumeration for details.

The third and fourth arguments are pointers to the `IIMGENC1_DynamicParams` and `IIMGENC1_Status` data structures respectively.

> **Note:**
>
> If you are using extended data structures, the third and fourth arguments must be pointers to the extended `DynamicParams` and `Status` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters.

‖ Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.

‖ Postconditions

The following conditions are true immediately after returning from this function.

- If the control operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise, it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- If the control command is not recognized, the return value from this operation is not equal to `IALG_EOK`.

‖ Example

See test application file, TestAppEncoder.c available in the \Client\Test\Src sub-directory.

‖ See Also

`algInit()`, `algActivate()`, `process()`

4.3.4 Data Processing API

Data processing API is used for processing the input data.

‖ Name

`algActivate()` – initialize scratch memory buffers prior to processing.

‖ Synopsis

```
Void algActivate(IALG_Handle handle);
```

‖ Arguments

```
IALG_Handle handle; /* algorithm instance handle */
```

‖ Return Value

```
Void
```

‖ Description

`algActivate()` initializes any of the instance’s scratch buffers using the persistent memory that is part of the algorithm’s instance object.

The first (and only) argument to `algActivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be initialized prior to calling any of the algorithm’s processing methods.

For more details, see *TMS320 DSP Algorithm Standard API Reference.* (literature number SPRU360).

‖ See Also

`algDeactivate()`

|| Name

`process()` – basic encoding/decoding call

|| Synopsis

```
XDAS_Int32 (*process)(IIMGENC1_Handle handle, XDM1_BufDesc
*inBufs, XDM1_BufDesc *outBufs, IIMGENC1_InArgs *inargs,
IIMGENC1_OutArgs *outargs);
```

|| Arguments

```
IIMGENC1_Handle handle; /* algorithm instance handle */

XDM1_BufDesc *inBufs; /* algorithm input buffer descriptor
*/

XDM1_BufDesc *outBufs; /* algorithm output buffer
descriptor */

IIMGENC1_InArgs *inargs /* algorithm runtime input
arguments */

IIMGENC1_OutArgs *outargs /* algorithm runtime output
arguments */
```

|| Return Value

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

|| Description

This function does the basic encoding/decoding. The first argument to `process()` is a handle to an algorithm instance.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `XDM_BufDesc` data structure for details).

The fourth argument is a pointer to the `IIMGENC1_InArgs` data structure that defines the run-time input arguments for an algorithm instance object.

The last argument is a pointer to the `IIMGENC1_OutArgs` data structure that defines the run-time output arguments for an algorithm instance object.

> **Note:**
>
> If you are using extended data structures, the fourth and fifth arguments must be pointers to the extended `InArgs` and `OutArgs` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters.

|| Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- `process()` can only be called after a successful return from `algInit()` and `algActivate()`.

- If algorithm uses DMA resources, `process()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.
- Buffer descriptor for input and output buffers must be valid.
- Input buffers must have valid input data.

‖ Postconditions

The following conditions are true immediately after returning from this function.

- If the process operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- After successful return from `process()` function, `algDeactivate()` can be called.

‖ Example

See test application file, TestAppEncoder.c available in the \Client\Test\Src sub-directory.

‖ See Also

`algInit()`, `algDeactivate()`, `control()`

|| Name

`algDeactivate()` – save all persistent data to non-scratch memory

|| Synopsis

```
Void algDeactivate(IALG_Handle handle);
```

|| Arguments

```
IALG_Handle handle; /* algorithm instance handle */
```

|| Return Value

```
Void
```

|| Description

`algDeactivate()` saves any persistent information to non-scratch buffers using the persistent memory that is part of the algorithm's instance object.

The first (and only) argument to `algDeactivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be saved prior to next cycle of `algActivate()` and processing.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

|| See Also

`algActivate()`

4.3.5 Termination API

Termination API is used to terminate the algorithm instance and free up the memory space that it uses.

|| Name

`algFree()` – determine the addresses of all memory buffers used by the algorithm

|| Synopsis

```
XDAS_Int32 algFree(IALG_Handle handle, IALG_MemRec
memTab[]);
```

|| Arguments

```
IALG_Handle handle; /* handle to the algorithm instance */

IALG_MemRec memTab[]; /* output array of memory records */
```

|| Return Value

```
XDAS_Int32; /* Number of buffers used by the algorithm */
```

|| Description

`algFree()` determines the addresses of all memory buffers used by the algorithm. The primary aim of doing so is to free up these memory regions after closing an instance of the algorithm.

The first argument to `algFree()` is a handle to the algorithm instance.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers previously allocated for the algorithm instance.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

|| See Also

`algAlloc()`

This page is intentionally left blank

Appendix A

Standard Huffman Table

This section provides an example for initialization of a structure of type `IDMJPGE_TIGEM_CustomHuffmanTables` using the Standard Huffman Table.

The sample code provided in this section is an example for initialization of a structure of type `IDMJPGE_TIGEM_CustomHuffmanTables` using the Standard Huffman Table.

Table A-1. Standard Huffman Table

```
#pragma
DATA_SECTION(JPEGENC_StandardHuffmanTable,".jpgenchufftab");

#pragma DATA_ALIGN(JPEGENC_StandardHuffmanTable,128);

const IDMJPGE_TIGEM_CustomHuffmanTables
JPEGENC_StandardHuffmanTable = {

/* VLC Tables */
/*Set 1 for Y Component */
/*DC-Y*/
/*const unsigned int JPEGENC_DCHUFFY[12] */
/*Length[16]-Codeword[16]*/

  0x00020000,

  0x00030002,

  0x00030003,

  0x00030004,

  0x00030005,

  0x00030006,

  0x0004000e,

  0x0005001e,

  0x0006003e,

  0x0007007e,

  0x000800fe,

  0x000901fe,

/*AC-Y*/

/*const unsigned int JPEGENC_ACHUFFY[16][16] */
```

```
/*Length[16]-Codeword[16]*/

0x0004000a, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x000b07f9,

0x00020000, 0x0004000c, 0x0005001c, 0x0006003a, 0x0006003b,
0x0007007a,

0x0007007b, 0x000800fa, 0x000901f8, 0x000901f9, 0x000901fa,
0x000a03f9,

0x000a03fa, 0x000b07f8, 0x0010ffeb, 0x0010fff5,

0x00020001, 0x0005001b, 0x000800f9, 0x000901f7, 0x000a03f8,
0x000b07f7,

0x000c0ff6, 0x000c0ff7, 0x000f7fc0, 0x0010ffbe, 0x0010ffc7,
0x0010ffd0,

0x0010ffd9, 0x0010ffe2, 0x0010ffec, 0x0010fff6,

0x00030004, 0x00070079, 0x000a03f7, 0x000c0ff5, 0x0010ff96,
0x0010ff9e,

0x0010ffa6, 0x0010ffae, 0x0010ffb6, 0x0010ffbf, 0x0010ffc8,
0x0010ffd1,

0x0010ffda, 0x0010ffe3, 0x0010ffed, 0x0010fff7,

0x0004000b, 0x000901f6, 0x000c0ff4, 0x0010ff8f, 0x0010ff97,
0x0010ff9f,

0x0010ffa7, 0x0010ffaf, 0x0010ffb7, 0x0010ffc0, 0x0010ffc9,
0x0010ffd2,

0x0010ffdb, 0x0010ffe4, 0x0010ffee, 0x0010fff8,

0x0005001a, 0x000b07f6, 0x0010ff89, 0x0010ff90, 0x0010ff98,
0x0010ffa0,

0x0010ffa8, 0x0010ffb0, 0x0010ffb8, 0x0010ffc1, 0x0010ffca,
0x0010ffd3,

0x0010ffdc, 0x0010ffe5, 0x0010ffef, 0x0010fff9,

0x00070078, 0x0010ff84, 0x0010ff8a, 0x0010ff91, 0x0010ff99,
0x0010ffa1,

0x0010ffa9, 0x0010ffb1, 0x0010ffb9, 0x0010ffc2, 0x0010ffcb,
0x0010ffd4,

0x0010ffdd, 0x0010ffe6, 0x0010fff0, 0x0010fffa,

0x000800f8, 0x0010ff85, 0x0010ff8b, 0x0010ff92, 0x0010ff9a,
0x0010ffa2,

0x0010ffaa, 0x0010ffb2, 0x0010ffba, 0x0010ffc3, 0x0010ffcc,
0x0010ffd5,

0x0010ffde, 0x0010ffe7, 0x0010fff1, 0x0010fffb,
```

```
0x000a03f6, 0x0010ff86, 0x0010ff8c, 0x0010ff93, 0x0010ff9b,
0x0010ffa3,

0x0010ffab, 0x0010ffb3, 0x0010ffbb, 0x0010ffc4, 0x0010ffcd,
0x0010ffd6,

0x0010ffdf, 0x0010ffe8, 0x0010fff2, 0x0010fffc,

0x0010ff82, 0x0010ff87, 0x0010ff8d, 0x0010ff94, 0x0010ff9c,
0x0010ffa4,

0x0010ffac, 0x0010ffb4, 0x0010ffbc, 0x0010ffc5, 0x0010ffce,
0x0010ffd7,

0x0010ffe0, 0x0010ffe9, 0x0010fff3, 0x0010fffd,

0x0010ff83, 0x0010ff88, 0x0010ff8e, 0x0010ff95, 0x0010ff9d,
0x0010ffa5,

0x0010ffad, 0x0010ffb5, 0x0010ffbd, 0x0010ffc6, 0x0010ffcf,
0x0010ffd8,

0x0010ffe1, 0x0010ffea, 0x0010fff4, 0x0010fffe,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

/*Set 2 for U & V Component */
```

```
/*DC-UV*/

/*const unsigned int JPEGENC_DCHUFFUV[12] */

/*Length[16]-Codeword[16]*/

  0x00020000,

  0x00020001,

  0x00020002,

  0x00030006,

  0x0004000e,

  0x0005001e,

  0x0006003e,

  0x0007007e,

  0x000800fe,

  0x000901fe,

  0x000a03fe,

  0x000b07fe,

/*AC-UV*/

/*const unsigned int JPEGENC_ACHUFFUV[16][16] */

/*Length[16]-Codeword[16]*/

0x00020000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x000a03fa,

0x00020001, 0x0004000b, 0x0005001a, 0x0005001b, 0x0006003a,
0x0006003b,

0x00070079, 0x0007007a, 0x000800f9, 0x000901f7, 0x000901f8,
0x000901f9,

0x000901fa, 0x000b07f9, 0x000e3fe0, 0x000f7fc3,

0x00030004, 0x00060039, 0x000800f7, 0x000800f8, 0x000901f6,
0x000a03f9,

0x000b07f7, 0x000b07f8, 0x0010ffb7, 0x0010ffc0, 0x0010ffc9,
0x0010ffd2,

0x0010ffdb, 0x0010ffe4, 0x0010ffed, 0x0010fff6,

0x0004000a, 0x000800f6, 0x000a03f7, 0x000a03f8, 0x0010ff97,
0x0010ff9f,

0x0010ffa7, 0x0010ffaf, 0x0010ffb8, 0x0010ffc1, 0x0010ffca,
0x0010ffd3,
```

```
0x0010ffdc, 0x0010ffe5, 0x0010ffee, 0x0010fff7,

0x00050018, 0x000901f5, 0x000c0ff6, 0x000c0ff7, 0x0010ff98,
0x0010ffa0,

0x0010ffa8, 0x0010ffb0, 0x0010ffb9, 0x0010ffc2, 0x0010ffcb,
0x0010ffd4,

0x0010ffdd, 0x0010ffe6, 0x0010ffef, 0x0010fff8,

0x00050019, 0x000b07f6, 0x000f7fc2, 0x0010ff91, 0x0010ff99,
0x0010ffa1,

0x0010ffa9, 0x0010ffb1, 0x0010ffba, 0x0010ffc3, 0x0010ffcc,
0x0010ffd5,

0x0010ffde, 0x0010ffe7, 0x0010fff0, 0x0010fff9,

0x00060038, 0x000c0ff5, 0x0010ff8c, 0x0010ff92, 0x0010ff9a,
0x0010ffa2,

0x0010ffaa, 0x0010ffb2, 0x0010ffbb, 0x0010ffc4, 0x0010ffcd,
0x0010ffd6,

0x0010ffdf, 0x0010ffe8, 0x0010fff1, 0x0010fffa,

0x00070078, 0x0010ff88, 0x0010ff8d, 0x0010ff93, 0x0010ff9b,
0x0010ffa3,

0x0010ffab, 0x0010ffb3, 0x0010ffbc, 0x0010ffc5, 0x0010ffce,
0x0010ffd7,

0x0010ffe0, 0x0010ffe9, 0x0010fff2, 0x0010fffb,

0x000901f4, 0x0010ff89, 0x0010ff8e, 0x0010ff94, 0x0010ff9c,
0x0010ffa4,

0x0010ffac, 0x0010ffb4, 0x0010ffbd, 0x0010ffc6, 0x0010ffcf,
0x0010ffd8,

0x0010ffe1, 0x0010ffea, 0x0010fff3, 0x0010fffc,

0x000a03f6, 0x0010ff8a, 0x0010ff8f, 0x0010ff95, 0x0010ff9d,
0x0010ffa5,

0x0010ffad, 0x0010ffb5, 0x0010ffbe, 0x0010ffc7, 0x0010ffd0,
0x0010ffd9,

0x0010ffe2, 0x0010ffeb, 0x0010fff4, 0x0010fffd,

0x000c0ff4, 0x0010ff8b, 0x0010ff90, 0x0010ff96, 0x0010ff9e,
0x0010ffa6,

0x0010ffae, 0x0010ffb6, 0x0010ffbf, 0x0010ffc8, 0x0010ffd1,
0x0010ffda,

0x0010ffe3, 0x0010ffec, 0x0010fff5, 0x0010fffe,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,
```

```
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000,

0x00000000, 0x00000000, 0x00000000, 0x00000000,

/* DHT Marker */

/* lum_dc_codelens */

  0, 1, 5, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,

/* lum_dc_ncodes */

  16,

/* lum_dc_symbols */

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,

/* lum_dc_nsymbols */

  12,

/* lum_ac_codelens */

  0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d,

/* lum_ac_ncodes */

  16,

/* lum_ac_symbols */

  0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,

  0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,

  0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,
```

```
  0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 0xf0,
  0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,
  0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,
  0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
  0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
  0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
  0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
  0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
  0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
  0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
  0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
  0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
  0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,
  0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
  0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,
  0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,
  0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
  0xf9, 0xfa,
/* lum_ac_nsymbols */
  162,
/* chm_dc_codelens */
  0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
/* chm_dc_ncodes */
  16,
/* chm_dc_symbols */
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
/* chm_dc_nsymbols */
  12,
/* chm_ac_codelens */
  0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77,
/* chm_ac_ncodes */
  16,
/* chm_ac_symbols */
  0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,
```

```
  0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,
  0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,
  0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33, 0x52, 0xf0,
  0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34,
  0xe1, 0x25, 0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26,
  0x27, 0x28, 0x29, 0x2a, 0x35, 0x36, 0x37, 0x38,
  0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
  0x49, 0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
  0x59, 0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
  0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
  0x79, 0x7a, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
  0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95, 0x96,
  0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5,
  0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4,
  0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3,
  0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2,
  0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda,
  0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,
  0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
  0xf9, 0xfa,
/* chm_ac_nsymbols */
  162
};
```

Appendix B

Standard Quantization Table

This section provides an example for initialization of a structure of type `IDMJPGE_TIGEM_CustomQuantTables` using the standard Quantization Table.

The sample code provided in this section is an example for initialization of a structure of type `IDMJPGE_TIGEM_CustomQuantTables` using the Standard Quantization Table.

Table B-1.Standard Quantization Table

```
const IDMJPGE_TIGEM_CustomQuantTables
JPEGENC_StandardQuantTable= {

  /* LUMA (Y) */

  16, 11, 10, 16, 24, 40, 51, 61,

  12, 12, 14, 19, 26, 58, 60, 55,

  14, 13, 16, 24, 40, 57, 69, 56,

  14, 17, 22, 29, 51, 87, 80, 62,

  18, 22, 37, 56, 68, 109, 103, 77,

  24, 35, 55, 64, 81, 104, 113, 92,

  49, 64, 78, 87, 103, 121, 120, 101,

  72, 92, 95, 98, 112, 100, 103, 99,


  /* CHROMA (Cb Cr) */

  17, 18, 24, 47, 99, 99, 99, 99,

  18, 21, 26, 66, 99, 99, 99, 99,

  24, 26, 56, 99, 99, 99, 99, 99,

  47, 66, 99, 99, 99, 99, 99, 99,

  99, 99, 99, 99, 99, 99, 99, 99,

  99, 99, 99, 99, 99, 99, 99, 99,

  99, 99, 99, 99, 99, 99, 99, 99,

  99, 99, 99, 99, 99, 99, 99, 99
};
```