# *xDAIS-DM (Digital Media) User Guide*

Literature Number: SPRUEC8B
January 2007

TEXAS INSTRUMENTS

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

# Preface

## *About This Manual*

This document describes the xDAIS-DM (Digital Media) standard that is built over TI's well proven eXpress DSP Algorithm Interoperability Standard (also known as xDAIS) specification.

This document refers to xDAIS-DM as xDM. The terms xDAIS-DM and xDM are used interchangeably in this document.

xDM defines a uniform set of APIs across various multimedia algorithms to ease integration and ensure interoperability.

## *Intended Audience*

This document assumes that you are fluent in the C language, have a good working knowledge of digital signal processing and the requirements of DSP applications, and have had some exposure to the principles and practices of object-oriented programming. This document describes the interfaces between algorithms and the applications that utilize these algorithms. System integrators will see how to incorporate multiple algorithms from separate developers into a complete system. Algorithm writers learn how to ensure that an algorithm can coexist with other algorithms in a single system and how to package an algorithm for deployment into a wide variety of systems.

## *How to Use This Manual*

This document contains the following chapters:

- **Chapter 1 – Overview**, provides an overview of the xDM standard and the generic interfaces defined by it.
- **Chapter 2 – Algorithm Interfaces**, contains the algorithm interfaces that are defined by xDM.

## Additional Documents and Resources

The following online system provides reference information about the API used to create xDM-compatible algorithms:

- *xDM API Reference.* XDAIS_INSTALL_DIR/docs/html/index.html

You can use the following books to supplement this user's guide:

- TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352)
- TMS320 DSP Algorithm Standard API Reference (SPRU360)
- TMS320 DSP Algorithm Standard Developer's Guide (SPRU424)
- TMS320 DSP Algorithm Standard Demonstration Application (SPRU361)

## Text Conventions

The following conventions are used in this specification:

- 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`.

# Contents

# Figures

Chapter 1

# Overview

This chapter provides an overview of the xDM standard.

**Topic** **Page**



The xDM standard defines a uniform set of APIs across various multimedia codecs to ease integration and ensure interoperability. xDM is built over TI's well proven eXpress DSP Algorithm Interoperability Standard (also known as xDAIS) specification.

## 1.1 Scope of the Standard

xDM addresses the following:

- Uniform lightweight APIs across various classes of multimedia algorithms, such as audio, video, speech, and image
- Flexibility of extension for various requirements such as metadata parsing, file format, custom processing, and so forth
- Interoperability across various algorithms and vendors

xDM does not address the following:

- Metadata parsing from multimedia streams
- File format or multiplex support
- Digital Rights Managements (DRM) interaction with codecs
- Call back from algorithms and applications to enable data movement and processing
- APIs other than codecs, for example, pre- and post-processing APIs like resizing, echo cancellations, and so forth

These features may be addressed in later versions of xDM (version 1.x, 2.x). The features are not standardized in xDM v1.0, the basic definition can be extended to support above mentioned features.

## 1.2 Goals of the Standard

The goals of this standard include:

- Enable plug and play architecture for multimedia codecs across various classes of algorithms and vendors.
- Enable faster time to market for multimedia products such as, digital cameras, cell phones, set-top boxes, and portable multimedia players.
- Provide a standard interface based on given class of multimedia codecs (for example, audio, video, image, and speech).
- Define common status and parameters based on given class of multimedia codecs.
- Flexibility of extension of custom functionality.
- Low overhead of interface.
- Reduce integration time for system developers.

## 1.3 xDM Interface History and Roadmap

The xDM 0.9 version was released with xDAIS 5.00. xDM 0.9 will continue to be provided and supported for the near term, but is now deprecated. Support for the 0.9 interfaces will be removed in the future.

The xDM 1.0 beta version was released with xDAIS 5.10. The xDM 1.0 final version is released with xDAIS 5.20. With this 1.0 final release, the 1.0 beta interfaces are no longer supported.

For details about differences between xDM versions 0.9 and 1.0 final, see XDAIS_INSTALL_DIR/packages/ti/xdais/dm/docs/xdm1_differences.pdf. The xdm1_differences.pdf file contains a list of changes that are likely to be needed in order to migrate your xDM 0.9-compliant algorithms to xDM 1.0-compliant algorithms.

## 1.4 Relationship Between xDM and xDAIS

xDM is an extension to the IALG interface standard. It defines and standardizes interfaces for multimedia codecs. The relationship between xDM and xDAIS is depicted in Figure 1-1.

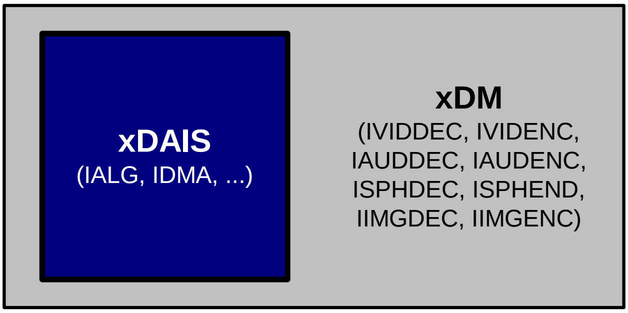


*Figure 1-1. Relationship between xDM and xDAIS*

As shown in the figure, xDM extends the xDAIS standard. xDM defines eight generic interfaces for the following categories. The "x" suffix represents a version of the interface. In xDM 0.9, the suffix was omitted; in xDM 1.0, it is "1".

- IVIDENCx - Generic interface for video encoders
- IVIDDECx - Generic interface for video decoders
- IAUDENCx - Generic interface for audio encoders
- IAUDDECx - Generic interface for audio decoders
- ISPHENCx - Generic interface for speech encoders
- ISPHDECx - Generic interface for speech decoders
- IIMGENCx - Generic interface for image encoders

- IIMGDECx - Generic interface for image decoders

### 1.4.1 xDAIS Interfaces

The current xDAIS interface from algorithm to application is depicted in Figure 1-2.

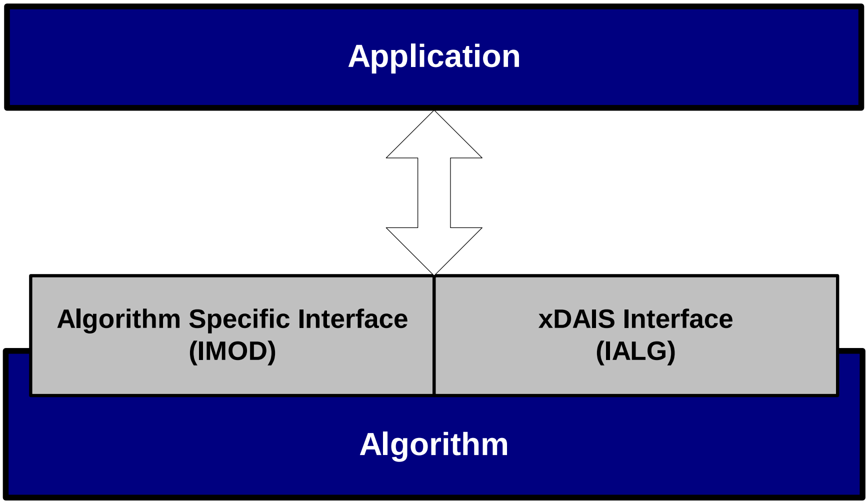


*Figure 1-2. Current xDAIS Interface to Codec Library*

As per xDAIS, the given algorithm extends the standard IALG interface to the IMOD interface (algorithm specific). The IMOD interface provides basic functionality of the algorithm, while the IALG interface takes care of the memory management. xDAIS does not define the IMOD interface. The algorithm implementer must define the IMOD interface based on his requirements. For example, in case of MP3, the algorithm will implement the IMP3 interface. This interface is kept totally open.

The application talks to the codec library via the IMOD interface. Optionally, an application can directly talk to the codec library via the MOD interface, which provides high level functionality.

The xDM standard adds a new interface as depicted in Figure 1-3.

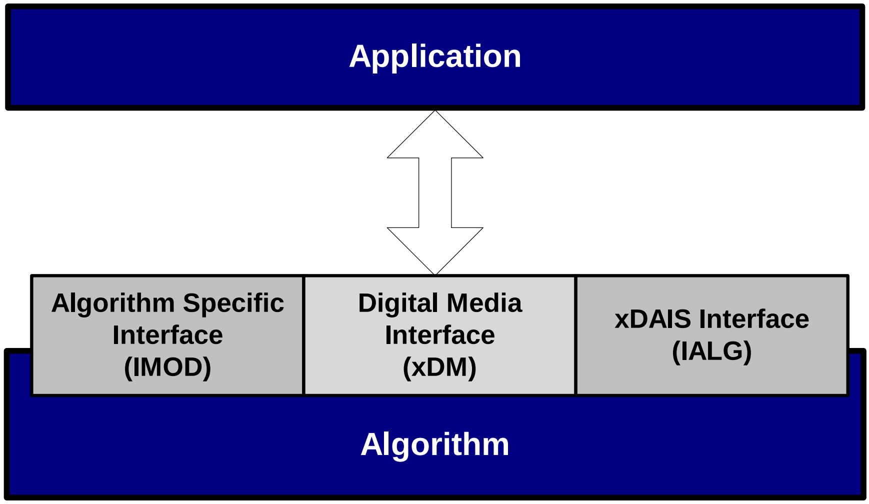


*Figure 1-3. xDM Interface Added to Codec Library*

The algorithm uses one of eight predefined standard interface called xDM. This interface is a superset of the IALG interface. You can tailor a given algorithm or implementation by extending the xDM interface to the IMOD interface. In simple cases, IMOD will be identical to xDM. Optionally, the algorithm can add more functionality to the xDM interface to define the IMOD interface. In this case, the application talks to the codec library via the IMOD interface as done previously.

Chapter 2

# Algorithm Interfaces

This chapter describes all the algorithm interfaces that are defined by the xDM standard. Reference information about the interfaces is provided at XDAIS_INSTALL_DIR/docs/html/index.html.

## 2.1 IALG Interface

IALG interface defines a framework independent interface for the creation of algorithm instance objects. For more information on the IALG interface, see *TMS320 DSP Algorithm Standard API Reference*.

## 2.2 xDM Interface

The xDM interface consists of eight interfaces to tailor to various multimedia algorithms (for example, audio, video, encoders, decoders, and so forth). Reference information about these interfaces is provided at in an online reference system at XDAIS_INSTALL_DIR/docs/html/index.html.

These new interfaces define two new functions:

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

The following figure summarizes the valid sequences of execution of the functions for a particular algorithm instance.

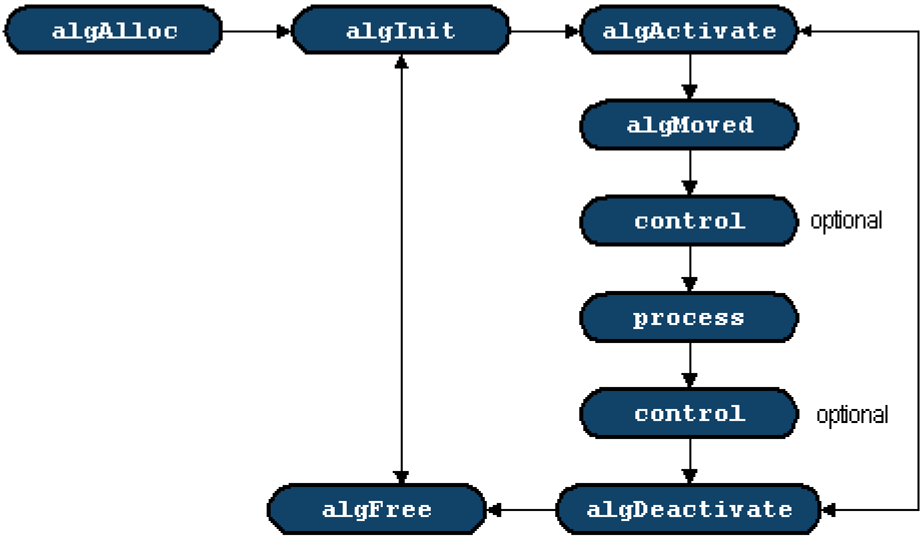


*Figure 2-1. xDM Interface Function Call Order.*

The xDM interface function call order is similar to the IALG interface function call order, except that xDM removes usage of the `algControl()` method. Hence when xDAIS is used along with xDM, instead of using the `algControl()` method (which is part of the IALG interface), the algorithm developer must use the `control()` method (which is part of the xDM interface).

## 2.3 Extending the xDM Interfaces

You can optionally tailor a given algorithm or implementation by extending the xDM interface to create a codec-specific IMOD interface. The algorithm can add more functionality to the xDM interface to define the IMOD interface as shown in Figure 2-2.

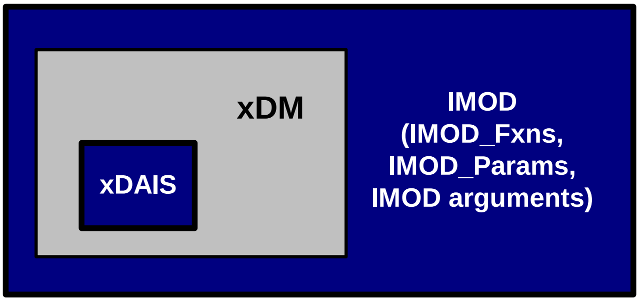


*Figure 2-2. xDM and IMOD Interfaces*

The relationship between the xDM and IMOD interfaces are as follows:

- **IMOD_Fxns.** xDM functions and extension functions
- **IMOD_Params.** xDM Params (consist of creation and run time) + extension parameters (creation and runtime)
- **IMOD arguments.** Includes IMOD_InArgs, IMOD_OutArgs, IMOD_DynamicParams, and IMOD_Status

Note that the fields in most of these structures changed from xDM version 0.9 to 1.0. See Section 1.3, “xDM Interface History and Roadmap”, for more information.

### 2.3.1 Extending an Algorithm

The ti.xdais.dm.examples.videnc1_copy example demonstrates how to extend VIDENC1_InArgs. The extended structure, which is from videnc1_copy_ti.h, is as follows:

```
typedef struct IVIDENC1CPY_InArgs {
   IVIDENC1_InArgs videnc1InArgs;
   XDAS_Int32      maxBytes;  /* Max # of bytes to copy */
} IVIDENC1CPY_InArgs;
```

The implementation of the process() function, which uses this optional field, is as follows in videnc1_copy.c:

```
/*
 *  ======== VIDENC1COPY_TI_process ========
 */
XDAS_Int32 VIDENC1COPY_TI_process(IVIDENC1_Handle h,
    XDM_BufDesc *inBufs, XDM_BufDesc *outBufs,
    IVIDENC1_InArgs *inArgs, IVIDENC1_OutArgs *outArgs)
{
    XDAS_Int32 numSamples;

    /* .... */

    /* there's an available in and out buffer, how many samples? */
    numSamples = inBufs->bufSizes[0] < outBufs->bufSizes[0] ?
         inBufs->bufSizes[0] : outBufs->bufSizes[0];

    /* honor the extended maxBytes if it was provided */
    if (inArgs->size == sizeof(IVIDENC1CPY_InArgs)) {
        if (numSamples > ((IVIDENC1CPY_InArgs *)inArgs)->maxBytes) {
            numSamples = ((IVIDENC1CPY_InArgs *)inArgs)->maxBytes;
        }
    }

    /* process the data: read input, produce output */
    memcpy(outBufs->bufs[0], inBufs->bufs[0], numSamples);

    /* ... */
}
```

### *2.3.2 Extension Considerations for Remotablity*

To enable extensions, most xDM structures contain size as their first field. This field is used:

- By the framework to determine how big the structure is.
- By the codec to determine how to interpret the fields.

Some frameworks may impose further constraints. For example, the Codec Engine, because it is RPC-based, has the following constraints when using remote codecs:

- **No pointers may be used in the extended fields.** Because of address translation (the GPP-side address doesn't match the DSP-side address) and cache maintenance (the DSP is cached, which requires maintenance for coherence), pointers to data are non-trivial to manage. The default VISA RPC stubs and skeletons manage pointers defined in the base class, but it's impossible for them to know about pointers in proprietary extensions.
- **Maximum size of 256 bytes.** In the default configuration, the messages used by the VISA stubs and skeletons to do IPC are 4 KB in size. The messages in the case of a single-processor DSP/BIOS application are 1 KB. These messages must contain internal headers (< 128 bytes), input structs (for example, InArgs or DynamicParams), and output structs (for example, OutArgs or Status).

Note that these particular constraints are specific to the Codec Engine framework. Other frameworks may impose different constraints; consult your framework's documentation for details.

### 2.3.3 Alignment and Structure Packing Considerations

xDM-compliant algorithms are intended to be executed in a variety of environments. One topic to consider is that of field alignment and packing within the interface structures. In a simple, single-processor environment, in which the application passes arguments directly to the algorithm and both are built with the same set of tools, alignment may seem an unnecessary concern.

However, if the application runs on a different processor than the algorithm, one processor may be byte-addressable while the other is not. In this case, tightly packed and/or byte-aligned structures could be problematic and could require more a complex framework to correctly pack and unpack these structures during runtime, potentially impacting performance. As a result, almost all xDM base structure fields are aligned on 32-bit boundaries.

**Note:** The xDM 1.0 speech interfaces are an exception. Some fields are aligned on 16-bit boundaries. This was a conscious decision, made to enable high-density systems at the potential expense of requiring more complex frameworks. Note, however, that all xDM structures *end* on a 32-bit boundary. This was done to prevent potential packing issues when these structures are extended.

Similar issues apply in the case of extending structures (see Section 2.3.1). We recommend that when you create custom interfaces, you use the same 32-bit alignment for extended structures. This makes alignment requirements easier on the application and/or the framework used between the application and algorithm. Extra padding may be used to ensure correct alignment as shown in the following example:

```
typedef struct MYIVIDDEC_DynamicParams {
    IVIDDEC_DynamicParams base;
    XDAS_Int32 newField1;
    XDAS_Int16 newField2;
    XDAS_Int16 padding;  // Needed for 32-bit alignment
} MYIVIDDEC_DynamicParams;
```

Chapter 3

# Glossary

**Algorithm**: Technically, an algorithm is a sequence of operations, each chosen from a finite set of well-defined operations (for example, computer instructions), that halts in a finite time, and computes a mathematical function.

**API**: Acronym for application programming interface. A specific set of constants, types, variables, and functions used to programmatically interact with a piece of software.

**Framework**: Part of an application that is designed to remain invariant while selected software components are added, removed, or modified. Very general frameworks are sometimes described as application-specific operating systems.

**Interface**: A set of related functions, types, constants, and variables. An interface is often specified with a C header file.

**Method**: A synonym for a function that is part of an interface.

**Module**: A module is an implementation of one (or more) interfaces. In addition, all modules follow certain design elements that are common to all xDM-compatible software components. Roughly speaking, a module is a C language implementation of a C++ class.