SMPTE RP 2073-2:2022
Revision of SMPTE RP 2073-2:2017

SMPTE RECOMMENDED PRACTICE

VC-5 Video Essence
Part 2. Conformance Specification

Table of Contents

Page

445 Hamilton Ave., White Plains NY 10601, (914) 761-1100
Approved February 8, 2022

Figures

Tables

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual. This SMPTE Engineering Document was prepared by Technology Committee 10E.

The following changes were made to RP 2073-2:2017 for this revision:

1. Added new section 9.4 defining metadata conformance testing for SMPTE ST 2073-7.
2. Added new Annex C describing how the test materials for metadata conformance testing were created.
3. Added new Annex D listing the input and output files to be used for each step of metadata conformance testing.
4. Editorial changes.

Intellectual Property

At the time of publication no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

Introduction

This section is entirely informative and does not form an integral part of this Engineering Document.

VC-5 is a wavelet-based intra-frame codec intended for applications that require fast decoding and encoding with high visual quality. The VC-5 codec is suitable for image capture and post production.

This document describes the test materials and procedures for verifying that:

1. A bitstream produced by an implementation of a VC-5 encoder is compliant with the VC-5 essence standard and
2. An implementation of a VC-5 decoder is compliant with the VC-5 essence standard.

1 Scope

The VC-5 essence standard comprises the SMPTE standards designated SMPTE ST 2073-1, ST 2073-3, ST 2073-4, ST 2073-5, ST 2073-6, and ST 2073-7.

This recommended practice specifies criteria and procedures for testing the conformance of encoder and decoder implementations to the VC-5 essence standard and describes the test materials used for conformance testing.

The test materials used for conformance testing comprise:

1. Reference decoder for testing bitstreams created by implementations of a VC-5 encoder for compliance with the VC-5 essence standard,

2. Bitstreams and reference images for testing implementations of a VC-5 decoder for compliance with the VC-5 essence standard.

Although not required for conformance testing, the test materials include a sample encoder, the images used by the sample encoder to create the bitstreams used for conformance testing, a program for converting between image formats, and a program for comparing image files. This recommended practice describes the image file formats used by the software and scripts that automate the conformance testing procedure.

Detailed instructions for installing and building the programs and contact information for submitting bug reports are provided in a separate document that is included in the software distribution of the test materials.

This document does not include the test materials which are located as specified in Annex E.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; Tables shall be next; then formal languages; then figures; and then any other language forms.

3 Normative References

There are no normative references in this document.

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1 base64

representation of binary data using printable characters as defined in IETF RFC 4648

4.2 big-endian

order of bytes that comprise a number with more significant bytes before less significant bytes

4.3 dark metadata

metadata that is not defined in any published standard

4.4 extrinsic metadata

metadata defined in a published standard other than SMPTE ST 2073-7

4.5 intrinsic metadata

metadata defined in SMPTE ST 2073-7:2022 Annex B

4.6 little-endian

order of bytes that comprise a number with less-significant bytes before more-significant bytes

4.7 FourCC

media format identifier consisting of four upper or lower case alphanumeric US-ASCII characters

4.8 format specification string

character string that specifies the format of the string output by the C language `sprintf` function

4.9 nested metadata

intrinsic metadata than can include nested tuples

4.10 specified parts

documents in the VC-5 essence standard selected for conformance testing

4.11 simple metadata

intrinsic metadata that does not contain nested tuples

4.12 streaming data

metadata defined in SMPTE ST 2073-7:2022 Annex C

4.13 VC-5 essence standard

SMPTE ST 2073-1, SMPTE RP 2073-2, SMPTE ST 2073-3, SMPTE ST 2073-4, SMPTE ST 2073-5, SMPTE ST 2073-6, SMPTE ST 2073-7, and SMPTE ST 2073-10

5 Overview (Informative)

This document specifies how to test conformance of an encoder or decoder implementation with the specified parts. The testing procedure can select which parts of the VC-5 essence standard to test for conformance.

The test materials provided as part of this conformance specification include:

1. Source code for a reference decoder,
2. Source code for a sample encoder,
3. Utility programs and scripts useful for testing VC-5 decoder and encoder implementations,
4. Reference bitstreams for testing conformance of a decoder implementation to the VC-5 essence standard,
5. Images used to encode the reference bitstreams,
6. Reference metadata for testing conformance with ST 2073-7.

The bitstream defined in the VC-5 essence standard represents an ordered set of component arrays. In typical practice, each component array corresponds to an image plane containing color values of the same type. An image unpacking process unpacks an input image into component arrays for input to the encoding process. The image unpacking process is not defined by the VC-5 essence standard. Likewise, a decoding process outputs an ordered set of component arrays that can be packed into an image by an image repacking process that is not defined by the VC-5 essence standard.

The sample encoder and reference decoder provided with the test materials include image unpacking and repacking code to enable testing with common image formats.

The test materials are described in section 6. The reference decoder and sample encoder are written in standard C programming language (section 6.1) and can test the combinations of the parts of the VC-5 standard (section 6.2).

The reference decoder for testing bitstreams created by implementations of a VC-5 encoder for compliance with the VC-5 essence standard is described in section 6.3. Reference bitstreams are provided for testing decoder compliance (Annex A).

The test materials include a sample encoder (section 6.4), a program for converting between image formats (section 6.5), a program for comparing image files (section 6.6), and the images used by the sample encoder to create the reference bitstreams used for conformance testing (Annex B).

The test materials include tools for creating and comparing metadata (section 6.7) and sample metadata in XML format (Annex C) as defined in SMPTE ST 2073-7:2022 Annex A.

Section 6.8.6 provides an overview of the tools and methods for verifying conformance. The procedure for verifying compliance with provisions of SMPTE ST 2073-7 is defined in section 9.4. Reference metadata and bitstreams for testing conformance to SMPTE ST 2073-7 are listed in Annex D.

The image file formats used by the sample encoder, reference decoder, and utility programs are described in section 7. Procedures and scripts that can be used for conformance testing are described in section 8. The criteria for conformance to the VC-5 essence standard are specified in section 9. Instructions for installing the test materials are provided in section 10 and in a text file provided with the test materials distribution.

Instructions for installing and building the programs and contact information for submitting bug reports are provided in a separate document that is included in the test materials distribution.

A VC-5 bitstream can represent common image formats as well as Color Filter Array (CFA) images such as Bayer. The component values represented in a VC-5 bitstream can have up to 24 bits of precision. To accommodate the variety of images that can be encoded into a VC-5 bitstream, the software provided with the test materials can use images stored as unformatted binary files (section 7.1) or an ordered set of component arrays stored as one array per file with the channel number encoded into the pathname (section 7.1.4). For convenience, the software also supports the DPX file format (section 7.1.5).

Reference bitstreams for testing VC-5 decoder implementations for compliance with the VC-5 essence standard and the corresponding reference images decoded from the reference bitstreams by the reference decoder are listed in Annex A.

6 Description of the Test Materials

6.1 Source Code

The source code for the reference decoder and sample encoder is written in the C programming language according to ISO/IEC 9899:2011.

Tools for creating test cases and running conformance tests are written in Python 3 (URL provided in the bibliography) and Tcl, described in the book Tcl and the Tk Toolkit listed in the bibliography.

For the purposes of this document, the symbol $(ROOT) is defined to be the root directory in which the test materials are located. See Annex E for further information.

6.2 Specified Parts

The specified parts for a build of the sample encoder or reference decoder are specified at compile-time and runtime as described in Table 1. An implementation of a VC-5 encoder or decoder shall be conformant with the specified parts.

The first column in Table 1 indicates which parts can be selected for conformance testing.

Table 1 — Parts of the VC-5 essence standard.

Enabled Part Number	VC-5 Essence Standard Title	Implied Part Numbers
1	ST 2073-1 Elementary Bitstream	None
3	ST 2073-3 Image Formats	1
4	ST 2073-4 Subsampled Color Difference Components	1, 3
5	ST 2073-5 Layers	1, 3
6	ST 2073-6 Sections	1, 3
7	ST 2073-7 Metadata	1

The reference decoder and sample encoder allow different parts of the VC-5 essence standard to be tested. Parts are enabled through a combination of compile-time and runtime variables.

The specified parts for conformance testing shall be enabled at both compile-time and runtime.

A part can be enabled at compile-time using the `VC5_ENABLED_PARTS` compile-time variable in the file `$(ROOT)/common/include/config.h` that is provided with the source code distribution. A part can be enabled at runtime by using the parts command-line argument to explicitly enable that part. Some parts are implicitly enabled at runtime when another part is enabled (see implied part numbers in Table 1).

Part 1 is always enabled and does not need to be specified on the command-line. Specifying part 2 has no effect as there is no code associated with the conformance specification. Part 4 implies support for part 3 since ST 2073-4 requires features specified in ST 2073-3.

Parts 5 and 6 imply support for part 3 since it is expected that most VC-5 implementations will always use the image format features specified in part 3. This behavior can be changed by modifying compile-time variables in the file `$(ROOT)/common/include/config.h` (see the documentation provided with the test materials).

An implementation of ST 2073-7 depends only on ST 2073-1 to verify conformance, but applications might depend on other parts to make use of the decoded metadata.

The reference decoder and sample encoder distributed with the test materials have parts 1, 3, 4, 5, 6 and 7 enabled at compile-time by default. If the reference decoder and sample encoder are built with all parts enabled, then the same build can be used for testing any part by enabling that part at runtime using the parts command-line argument.

To encode a bitstream that includes syntax elements defined in VC-5 parts 3 and 4, run the encoder as follows:

```
encoder --parts 4 [other options] <image file> <bitstream file>
```

In this document, when the text says that a VC-5 part is enabled, it means that the part is enabled at both compile-time and runtime using the variables and command-line options described above.

6.3 Reference Decoder

6.3.1 General Usage

The reference decoder reads a bitstream that is compliant with the VC-5 essence standard and outputs one or more images. Each image is output as an unformatted file (section 7.1.1), an ordered set of component array files (section 7.1.4), or a DPX file with 10-bit packed RGB color values (section 7.1.5). The reference decoder includes an implementation of an image repacking process that packs the component arrays output by the decoder into an output image.

To invoke the reference decoder from the command line:

```
decoder [options] <bitstream file> <image file>
```

Command-line options for the reference decoder are listed in Table 2.

All options present on the command line come before the input bitstream file on the command line.

The reference decoder provided with the test materials includes an implementation of an image repacking process to allow the decoder to output packed images. The packed image file formats that are supported by the reference decoder are listed in Table 8. The code for the image repacking process can be modified to support other image formats.

Table 2 — Reference decoder command-line options.

Command Line Option (Short or Long Format)	Description
`-p <file format>` `--pixel <file format>`	File format of the image input to the sample encoder to enable the decoder to output an image in the same format as input to the sample encoder (default is the file format of the output image)
`-o <file format>` `--output <file format>`	File format of the output image (default is the file format corresponding to the filename extension of the output image file)
`-w <image width>` `--width <image width>`	Width of the input image in samples (default is the width obtained from the bitstream)
`-h <image height>` `--height <image height>`	Height of the input image in sample rows (default is the height obtained from the bitstream)
`-v` `--verbose`	Enable verbose output during decoding (default is no verbose output)
`-h` `--help`	Print the program usage and command-line options (default is not to print the program usage text)
`-P <parts list>` `--parts <parts list>`	List of part numbers from Table 1, separated by commas without spaces, to enable the corresponding parts of the decoder (section 6.2)
`-S <sections list>` `--sections <sections list>`	List of section numbers listed in Table 4, separated by commas, to enable processing of the specified sections during decoding

The value of `<file format>` is case insensitive and can be any of the file formats listed in Table 8. The value of `<image width>` or `<image height>` can be any unsigned integer that corresponds to valid dimensions of the component arrays produced by the image unpacking process. For example, the minimum width or height of a component array is 48, which implies that the minimum width and height of an image in BYR4 or NV12 format is 96.

6.3.2 Layer Decoding

The image file command-line argument can be a string format specification generated a sequence of output filenames so that each decoded layer is stored in a separate file. For example, if the output image file is `output-%04d.byr4` and the bitstream contains 4 layers, then the reference decoder outputs the layers in separate files as follows:

```
output-0001.byr4

output-0002.byr4

output-0003.byr4

output-0004.byr4
```

where `%04d` is replaced by the layer number.

If part 5 is enabled, then the reference decoder will output one image file per layer present in the bitstream using the image file provided on the command-line as a format specification string as described above.

If part 5 is not enabled, then the reference decoder will output the first layer present in the bitstream, storing the image in the filename specified by the image file provided as a command-line parameter, and ignore any remaining layers in the bitstream.

6.3.3 Section Decoding

If part 6 is enabled, then the reference decoder will output the type and size of each section specified by the sections command-line option that is encountered in the bitstream to the sections log file. The filename of the sections log file is the filename of the input bitstream file with the file extension replaced by "`log`".

If part 6 is not enabled, the reference decoder will ignore sections encountered in the bitstream.

Decoding is not affected by enabling or disabling sections, except for image sections as described in section 6.3.4.

6.3.4 Image Section Decoding

If part 6 is enabled and the sections command-line option specifies that image sections (section number 1) are to be processed, then the command-line arguments can include multiple output pathnames:

```
decoder [options] <bitstream file> <image file 1> <image file 2> … <image file n>
```

All command-line options precede the bitstream file command-line parameter. The bitstream contains the bitstream to be decoded. The remaining command-line arguments specify the output files for the decoded images. Output pathnames are used in the order in which image sections are present in the bitstream.

Typically, one image pathname is specified per image section in the bitstream, but the last output pathname can be a format specification string that generates pathnames for the output images. Only the last image pathname provided as a command-line argument can be a format string specification.

Providing a format specification string allows the decoder to output one image file for each image section encountered in the bitstream. If the last image pathname is not a format string specification, then the decoder will report an error if the number of image sections in the bitstream exceeds the number of image pathnames provided as command-line arguments.

6.4 Sample Encoder

6.4.1 General Usage

The sample encoder takes as input a single image that is either an unformatted file (section 7.1.1) or a DPX file with 10-bit packed RGB color values (section 7.1.5) and outputs a VC-5 compliant bitstream. The sample encoder includes an implementation of an image unpacking process for unpacking the input image into component arrays.

To invoke the sample encoder from the command line:

```
encoder [options] <image file> <bitstream file>
```

All options present on the command line come before the input image file on the command line.

Command-line options for the sample encoder are listed in Table 3. The image file can be one of the packed image file formats listed in Table 8. The encoded bitstream is placed in a binary file with the filename extension "`vc5`".

Table 3 — Sample encoder command-line options.

Command Line Option (Short or Long Format)	Description
`-w <image width>` `--width <image width>`	Width of the image in samples (needed when the file format does not provide the width)
`-h <image height>` `--height <image height>`	Height of the image in sample rows (needed when the file format does not provide the height)
`-p <file format>` `--pixel <file format>`	File format of the image (default is the file format corresponding to the filename extension of the image file)
`-q Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9`	Quantization divisors for the highpass subbands (default values are compiled into the encoder)
`-v` `--verbose`	Enable verbose output during encoding (default is no verbose output)
`-h` `--help`	Print the program usage and command-line options (default is not to print the program usage text)
`-P <parts list>` `--parts <parts list>`	List of part numbers from Table 1 separated by commas without spaces, to enable the corresponding parts of the encoder (see section 6.2)
`-S <sections list>` `--sections <sections list>`	List of section numbers from Table 4, separated by commas without spaces, specifying the types of sections defined in ST 2073-6 that are encoded in the bitstream
`-L <image section layers>` `--layers <image section layers>`	Number of layers per image section separated by commas without spaces (see section 6.4.5)

The unformatted image file formats (section 7.1.1) do not provide the image dimensions so the width and height are specified as command-line options. The file format can be inferred from the filename extension listed in Table 8 or passed as a command-line option. The sample encoder does not support using an ordered set of component arrays as an input format (section 7.1.4). A file format specified explicitly as a command-line option takes precedence over a file format inferred from the filename extension.

6.4.2 Layer Encoding

The default value for the layer count is 1.

If part 5 is enabled, then the sample encoder will encode input provided as command-line arguments, one filename per image into the VC-5 bitstream as defined in SMPTE ST 2073-5.

If part 5 is not enabled, then the sample encoder will encode the one image file provided as a command-line argument into the bitstream without adding any of the bitstream elements defined by SMPTE ST 2073-5.

6.4.3 Section Encoding

If part 6 is enabled, then

> If the sections command-line option is specified on the command line, then the sample encoder will insert the specified section elements into the bitstream for every section type defined in SMPTE ST 2073-6 and listed in Table 4.

If part 6 is not enabled, then

> If the sections command-line option is specified on the command line, then the sample encoder will report an error and not attempt to encode the bitstream.

Table 4 — Section type numbers and the corresponding section types.

Number	Section
1	Section element marking the beginning and extent of the portion of the bitstream containing the representation of an image
2	Section element marking the beginning and extent of the tag-value pairs that comprise the bitstream header
3	Section element marking the beginning and extent of the portion of the bitstream containing all syntax elements that are unique to a single layer (includes all channels in the layer)
4	Section element marking the beginning and extent of the portion of the bitstream containing all syntax elements that are unique to a single channel (includes all wavelets in the channel)
5	Section element marking the beginning and extent of the portion of the bitstream containing syntax elements that are unique to a single wavelet (includes all subbands in the wavelet)
6	Section element marking the beginning and extent of the portion of the bitstream containing tag-value pairs and the codeblock for a single subband

6.4.4 Image Section Encoding

If part 6 is enabled and the sections command-line option specifies that image sections (section number 1) are to be processed, then the command-line arguments can include multiple input pathnames:

```
encoder [options] <image file 1> <image file 2> … <image file n> <bitstream file>
```

Each image in each file will be encoded as an image section in the order in which the image files are listed on the command line. The image file pathnames cannot be format specification strings. The last command-line argument is the output file for the encoded bitstream.

6.4.5 Encoding Image Sections with Layers

Image sections can contain layers within the image section. Each layer image within an image section has the same image dimensions, pixel format, and image encoding. The number of input images specified on the command line equals the total number of layers in all image sections. The partitioning of input images into layers and sections is specified by the layers command-line option using a comma-separated list (no spaces) of the number of layers in each image section. For example, encode two image sections, the first image section with 2 layers and a second image section with 3 layers, using the command

```
encoder -P 5,6 -S 1 -L 2,3 image1a image1b image2a image2b image2c bitstream.vc5
```

Both parts 5 and 6 (layers and sections) are enabled, image sections are enabled, and the number of images provided on the command-line equals the total number of layers specified by the layers option.

6.5 Image Converter

The image converter is a utility program that is provided as a convenience. The program can convert between several of the image formats used by the test materials.

To invoke the converter from the command line:

```
converter [options] <input image file> <output image file>
```

The converter command-line options are described in Table 5.

The file formats of the input and output images are inferred from the filename extensions, but can be overridden using command–line options.

Table 5 — Image converter command-line options.

Option (Short or Long Format)	Description
`-w <image width>` `--width <image width>`	Width of the input image in samples (needed when the file format does not provide the width)
`-h <image height>` `--height <image height>`	Height of the input image in sample rows (needed when the file format does not provide the height)
`-p <file format>` `--pixel <file format>`	File format of the input image (default is the file format inferred from the input filename extension)
`-o <file format>` `--output <file format>`	File format of the output image (default is the file format inferred from the output filename extension)
`-v` `--verbose`	Enable verbose output during conversion (default is no verbose output)
`-h` `--help`	Print the program usage and command-line options (default is not to print the program usage text)

6.6 Image Comparer

The image comparer is a C language utility program that is provided as a convenience. The program computes the difference between two images.

To invoke the comparer from the command line:

```
comparer [options] <image file 1> <image file 2>
```

The comparer command-line options are described in Table 6.

Table 6 — Image comparer command-line options.

Option (Short or Long Format)	Description
`-w <image width>` `--width <image width>`	Width of both images in samples (required if the file format does not provide the width)
`-h <image height>` `--height <image height>`	Height of both images in sample rows (required if the file format does not provide the height)
`-p <file format>` `--pixel <file format>`	File format of both images (default is the filename extension)
`-v` `--verbose`	Enable verbose output during conversion (default is no verbose output)
`-h` `--help`	Print the program usage and command-line options (default is not to print the program usage text)

Both image files have the same image dimensions and pixel format.

6.7 Metadata Tools

The test materials include several programs written in C and Python as listed in Table 7.

The base64 format is defined in IETF RFC 4648.

Table 7 — Metadata tools.

Filename	Location	Description
`generate.py`	`$(ROOT)/metadata/python`	Tool for generating XML files of test data
`verify.py`	`$(ROOT)/metadata/python`	Tool for checking the correctness of metadata in XML format (SMPTE ST 2073-7:2022 Annex A)
`dumper.py`	`$(ROOT)/metadata/python`	Tool for extracting metadata chunk elements from a VC-5 bitstream
`compare.py`	`$(ROOT)/metadata/python`	Tool for comparing two files of XML test data for equivalence as defined in section 9.4.4
`parser.c`	`$(ROOT)/metadata/parser`	Tool for creating binary metadata from for the XML format as defined in SMPTE ST 2073-7:2022 Annex A

Filename	Location	Description
dumper.c	$(ROOT)/metadata/dumper	Tool for extracting metadata from VC-5 bitstreams as a binary file or in base64 or XML format as defined in SMPTE ST 2073-7:2022 Annex A
gpmf-parser	https://github.com/gopro/gpmf-parser.git	Tool for extracting streaming data in CSV format from an MP4 file
dpxdump.c	$(ROOT)/metadata/tools	Tool for extracting the DPX file header in base64 format
exrdump.c	$(ROOT)/metadata/tools	Tool for extracting the RGBA and CDCI essence descriptors from ACES containers in OpenEXR format
bintool.c	$(ROOT)/metadata/tools	Tool for converting between representations of binary data
dmcvt.py	$(ROOT)/metadata/python	Tool for generating DMCVT extrinsic metadata test cases in JSON format from an input file in base64 format
fileinfo.py	$(ROOT)/metadata/python	Tool for creating and updating the database of file information used for optional elements in extrinsic metadata
darken.py	$(ROOT)/metadata/python	Tool for creating dark metadata samples
xmpdump.py	$(ROOT)/metadata/python	Tool for extracting XMP metadata from a test case for comparing with the original XML metadata
update.py	$(ROOT)/metadata/python	Tool for updating the metadata test cases from working copies
verify.py	$(ROOT)/metadata/python	Tool for comparing the metadata test cases in two directory trees
testing.py	$(ROOT)/metadata/python	Tool for creating all metadata test cases using the generate.py script to measure code coverage
listfiles.py	$(ROOT)/metadata/python	Tool for listing all metadata test cases
beautify.py	$(ROOT)/metadata/python	Tool to output metadata test case payloads in a readable format

Detailed instructions for building and running the tools are included in the file `install.pdf` and `install.html` distributed in the test materials.

6.8 Test Cases

6.8.1 Testing Overview

The test materials include images organized into the directory structure specified in section 10.5.

Reference images for three test cases are provided in the test materials distribution: boxes, solid, and gradient. The boxes test case consists of synthetic images of randomly placed, overlapping rectangles with randomly chosen colors. The solid test case consists of synthetic images that are a single randomly chosen color. The gradient test case consists of synthetic images that are a single gray wedge from black to white.

The reference images that have width or height equal to 48 (or equal to 96 in the case of BYR4 and NV12) were chosen so that the component arrays output by the image unpacking process have the minimum width or height allowed by the VC-5 essence standard.

The reference images that have width or height equal to 49 (or equal to 97 in the case of BYR4 and NV12) were chosen so that the component arrays output by the image unpacking process have width or height that is an odd number. Wavelet transforms normally require an even input dimension, but SMPTE ST 2073-1 defined a mechanism for handling an odd width or height. The mechanism adds one to an odd dimension. In the case where the input width or height of the unpacked component arrays is 49, the corresponding dimension of each wavelet in the three level wavelet sequence will be 25, 13, and 7, respectively, so that the ability of a VC-5 encoder or decoder to handle odd wavelet dimensions will be exercised at each wavelet level.

6.8.2 Testing Layers

To enable testing of encoders that implement VC-5 Part 5 Layers, for each reference image listed in Section 6.8 an image is provided with the sequence number 0001, forming a pair of images that can be encoded as two layers in a single bitstream.

6.8.3 Testing Sections

Except for image sections (described in section 6.8.4), sections can be tested as part of testing any other part of the VC-5 standard by specifying part 6 on the command line (using the `-p` command-line option) along with the sections that are to be processed during decoding (using the `-s` command-line option)

If sections are enabled during decoding, then the decoder writes a log file containing information about the sections encountered while parsing the bitstream. The filename of the sections log file is the same as the filename of the input bitstream, with the file extension replaced by "`log`". The sections log file is written to the same directory as the files containing the decoded images.

6.8.4 Testing Image Sections

A different folder structure is used for the reference bitstreams and image files for testing image sections. The `sections` directory is a sub-directory of the `media` sub-directory of the root directory into which the codec software was installed (see section 10). Each subdirectory of the `sections` directory corresponds to an image sections test and contains the images files encoded as image sections into a single bitstream as part of the test.

Encoded bitstream files and decoded image files are written into the `results` sub-directory of the `sections` sub-directory.

6.8.5 Testing Image Section Layers

A different folder structure is used for the reference bitstreams and image files for testing image sections that contain nested layers. The layers directory is a sub-directory of the sections directory (see section 6.8.4). The layers directory contains images that are input to the sample encoder to create a reference bitstream, the reference bitstream produced by the sample encoder, and the reference images produced by the reference decoder from the reference bitstream.

6.8.6 Testing Metadata Conformance

SMPTE ST 2073-7 defines a mechanism for embedding metadata in the VC-5 bitstream. Metadata items are represented as tuples (SMPTE ST 2073-7:2022, section 7). Each metadata tuple begins with a unique four-byte tag (FourCC) that identifies the type of metadata. Metadata tuples can be nested within metadata tuples.

All metadata tuples are placed in a metadata chunk element (SMPTE ST 2073-7:2022, section 9). Metadata chunk elements have a unique tag that signals that the chunk contains metadata tuples and nothing more. Metadata can assist a decoder implementation in outputting a decoded image with specific characteristics, such as the color space, but metadata is not required to decode the image represented in the bitstream. A decoder implementation is not required to use metadata and can skip metadata chunks.

Since metadata is a separate portion of the bitstream, conformance to SMPTE ST 2073-7 can be verified by checking that an encoder implementation has embedded metadata in the bitstream using the defined chunk syntax, the metadata items in each metadata chunk satisfy the specifications as defined in SMPTE ST 2073-7:2022, the decoder implementation can extract the metadata items from metadata chunks in the bitstream, and the extracted metadata is equivalent (as defined in section 9.4.4) to the metadata embedded in the bitstream. How a decoder implementation uses the metadata is out of scope.

The encoder conformance testing workflows are diagramed in Figure 1. The sample encoder is used to create a reference bitstream for each metadata test case using one of the reference images listed in Annex B. The choice of the reference image does not matter since conformance to SMPTE ST 2073-7 only depends on the metadata. The encoded metadata extracted from the bitstream is used to verify compliance of an encoder implementation with the standard.

The decoder conformance testing workflows are diagramed in Figure 2. The reference bitstreams created by the sample encoder (described in the paragraph above and Figure 1) are used by the reference decoder to create a representation of the metadata without duplicates to satisfy metadata equivalence (defined in section 9.4.4). A decoder implementation is expected to produce an XML representation of the metadata in the bitstream that can be checked against the metadata from the reference decoder using the compare tool (section 6.7) or a functionally equivalent implementation of equivalence as defined in section 9.4.4.

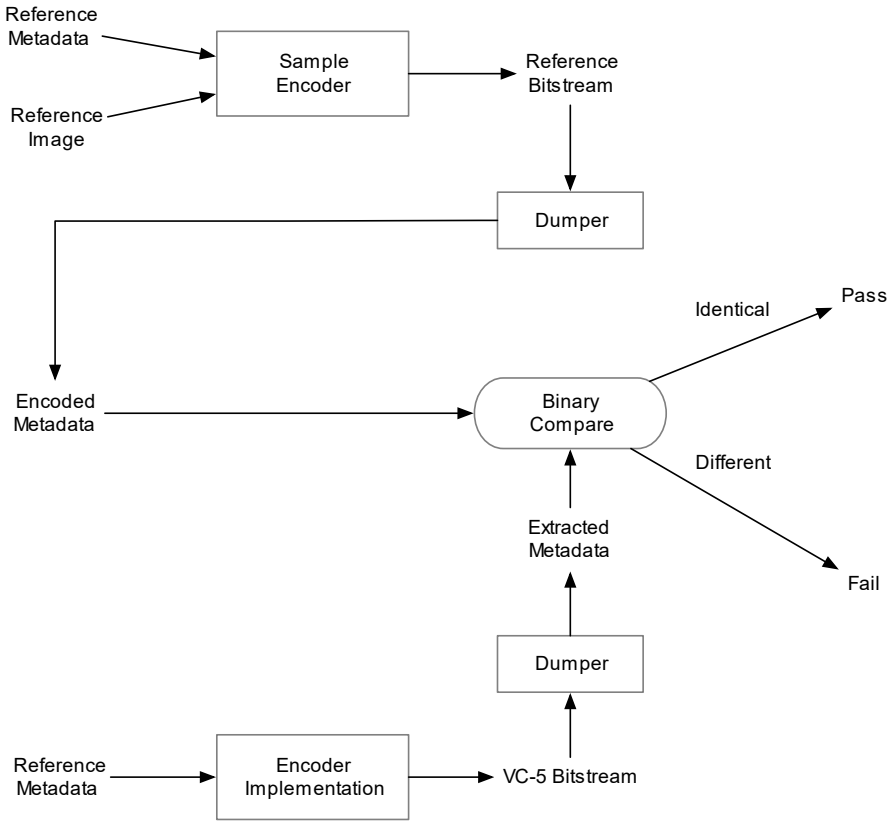

Figure 1 — Metadata conformance testing for encoder implementations.

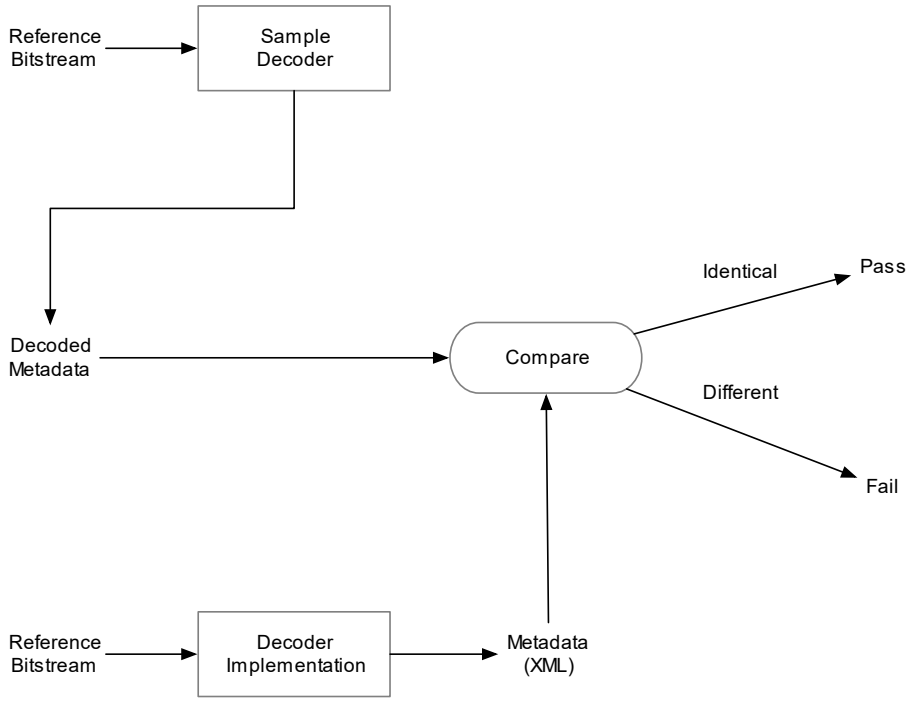

Figure 2 — Metadata conformance testing for decoder implementations.

7 File Formats

7.1 Image File Formats

7.1.1 Image File Formats Overview

The layout of an image in a file used for conformance testing can be described using sample arrays (Figure 3). The file format specifies the order of the component values, the number of bits per component value, and any padding between component values.

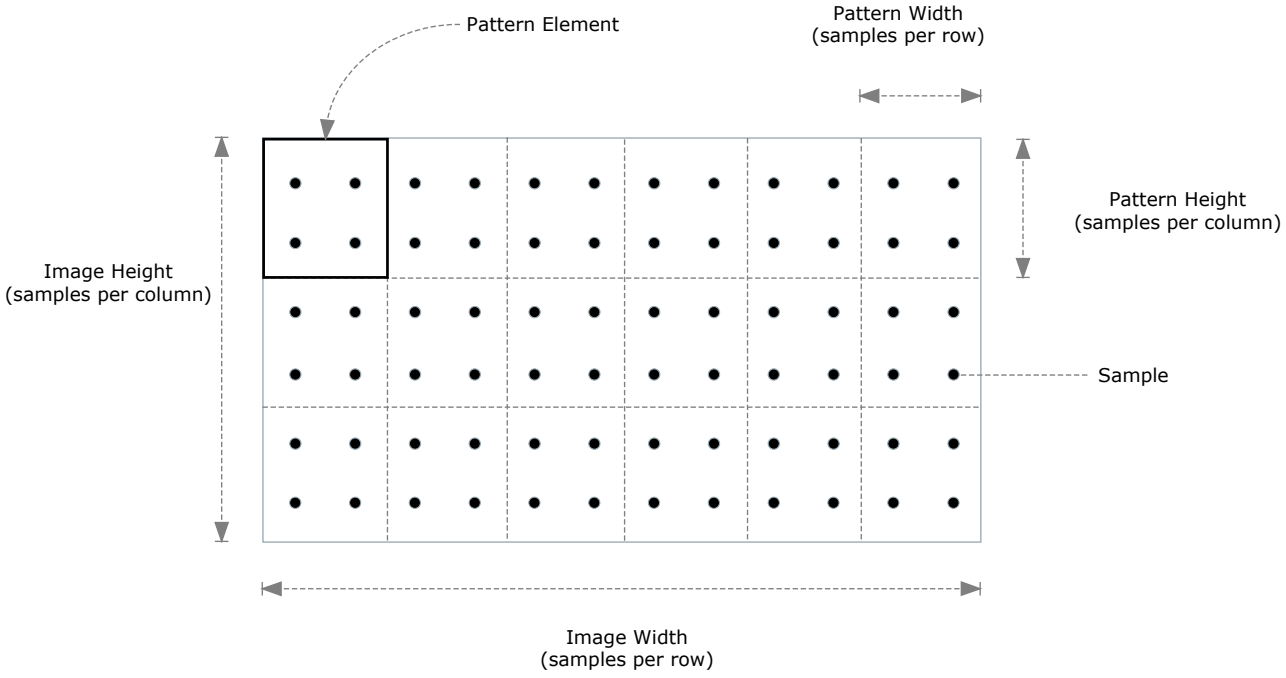

Figure 3 — Sample array.

By default, the file extension indicates the file format of the image file. The file extension can be lower case, upper case, or mixed case. The mapping from the lowercase representation of the file extension to the file format is summarized in Table 8.

Table 8 — File formats used by the test materials.

File Format	Description
`byr4`	Bayer pattern RGGB with 16 bits per color value, interleaved, each little-endian value in the range from 0 to 65535 inclusive (section 7.1.2)
`rg48`	R, G, B color components in order with 16 bits per color value, interleaved, each little-endian value in the range from 0 to 65535 inclusive (section 7.1.3)
`b64a`	A, R, G, B color components in order with 16 bits per color value, interleaved, each big-endian value in the range from 0 to 65535 inclusive (section 7.1.3)
`ca32`	Ordered set of component array files, one file per channel. Each component array file consists of the component array values stored as 32-bit unsigned integers in little-endian. The pathname is a valid string format specification for inserting the channel number (section 7.1.4).

File Format	Description
`dpx`	Image file compliant with SMPTE ST 268-1 and containing 10-bit RGB color values packed into a 32-bit word (section 7.1.5)
`nv12`	An array of Y', C'_B, and C'_R color components with 8 bits per color value and 4:2:0 color difference component subsampling as defined in ISO/IEC 13818-2:2013 (section 7.1.6)

With the exception of the DPX file format, all file formats listed in Table 8 are unformatted image files.

An unformatted image file contains the image samples stored in raster-scan order without any header. The number of samples per row is the width of the image without padding and the number of rows of samples is the height of the image without padding.

Except for the `ca32` and `nv12` image file formats, the pixel components in each unformatted image file format listed in Table 8 are interleaved. The `ca32` image file format stores each plane in a separate file.

All image files used by the programs described in this document store the image in the file as an array of samples in raster-scan order.

7.1.2 Bayer Images

Bayer images (pixel format `byr4`) can be described using a single sample array with pattern elements as defined in SMPTE ST 2073-3, sections 7 and 9.4, and Figure 3 in this document. The image data for the `byr4` file format comprises a single sample array using a 2 by 2 pattern element with the RGGB samples arranged in each pattern element as

R G
G B

Even-numbered rows of a Bayer image sample array comprise interleaved R and G color components, odd-numbered rows comprise interleaved G and B color components, and the first row of the sample array is row 0 (even). Each sample in the Bayer image sample array is stored in the file with 16 bits per color value, little-endian, in the range from 0 to 65535 inclusive.

7.1.3 RGB(A) Images

Other images with 4:4:4(:4) sampling such as R'G'B'(A) can be represented as sample arrays with a 1 by 1 pattern element.

The `rg48` pixel format stores R, G, and B color components in order with 16 bits per color value, interleaved, each little-endian value in the range from 0 to 65535 inclusive.

The `b64a` pixel format stores A, R, G, and B color components in order with 16 bits per color value, interleaved, each big-endian value in the range from 0 to 65535 inclusive.

7.1.4 Component Array Files

The `ca32` pixel format stores the internal representation of unpacked image planes used in the sample encoder and reference decoder with one image plane per file.

The reference decoder can output an ordered set of component arrays with one component array per file. Each file contains the values of the component array as 32-bit unsigned integers in raster-scan order with no image header. The component values are little-endian. The output pathname is a valid string format specification that allows the reference decoder to output multiple component array files, one per channel, with the channel number represented in the pathname. (See the documentation for the `sprintf` function in the standard C library). For example, if four component arrays are represented in the input bitstream and the output pathname is

```
boxes-%02d.ca32
```

then the reference decoder will output four files:

```
boxes-00.ca32

boxes-01.ca32

boxes-02.ca32

boxes-03.ca32
```

The dimensions of each component array are not represented in the unformatted component array files but would have been passed to the reference decoder as command-line arguments.

7.1.5 DPX Image Files

The DPX image format used for conformance testing is 10-bit RGB color values packed in a 32-bit word as indicated by the descriptor in the Image Information Header having the value 50. See SMPTE ST 268-1.

7.1.6 Subsampled YCbCr

Image data for the `nv12` file format comprises a sample array for the 8-bit Y' values immediately followed by a sample array with interleaved 8-bit C'_B and C'_R values. The sample array of $C'_BC'_R$ values is the same width as the Y' sample array and half the height of the Y' sample array. This file format implements 4:2:0 color difference component image subsampling as defined in See ISO/IEC 13818-2:2013 and SMPTE ST 2073-4.

7.1.7 Other Image File Formats

The sample encoder includes a subroutine called `ImageUnpackingProcess()` that can be modified to accept other input image formats and the reference decoder includes a subroutine called `ImageRepackingProcess`() that can be modified to produce other output image formats from the decoded component arrays.

7.1.8 Image File Naming Conventions

All image files in the test materials use a naming convention that specifies the image dimensions and pixel format in the filename itself. Since most image files are unformatted, the filename could be parsed to determine the image dimensions and pixel format, but the conformance testing scripts parse the directory hierarchy.

Image files in the test materials are organized into a directory hierarchy that identifies the image dimensions and pixel format as described in section 10.5.

The scripts that implement conformance testing traverse the test case directory tree and use the image directory name and pixel format directory name to pass the image dimensions and pixel format as command-line arguments to the reference decoder and sample encoder.

For example, when a script traverses the test case directory tree and encounters the image file

```
$(ROOT)/media/boxes/1280x720/byr4/boxes-1280x720-0000.byr4
```

the script has already visited the `1280x720` directory and determined the image dimensions by parsing the directory name (1280 columns by 720 rows) and already visited the `byr4` directory and determined the pixel format.

7.2 Bitstream File Formats

Files that contain VC-5 encoded bitstreams follow the naming conventions for image files with the exception that the part of VC-5 that was enabled is included in the pathname.

8 Conformance Testing Scripts (Informative)

8.1 Conformance Testing Overview

Conformance testing is performed using scripts that apply the sample encoder or reference decoder to all images in the test case directory tree described in section 8.2.

The scripts can be modified to test other implementations of the VC-5 codec. Additional user-generated test cases can be added to the test suite by adding image files to the appropriate directory in the directory tree of test cases.

8.2 Test Case Directory Tree

The root of the test case directory tree is the `$(ROOT)/media` directory. The first level in the test case directory tree is the name of the test case (see section 10.5). The second level in the test case directory tree is the image dimensions represented by the string “<width>x<height>”, for example 1280x480. The third level in the test case directory tree is the pixel format.

8.3 Codec Test Script

The scripts directory contains scripts written in Tcl that invoke the sample encoder and reference decoder on images in the media directory. The Tcl test script is in the file `testcodec.tcl` in the directory `$(ROOT)/scripts`.

The media directory is organized into a hierarchy that follows naming conventions. The Tcl test script relies on these naming conventions. The media directory contains one subdirectory per test case. The directory for each test case contains one subdirectory for each unique image width and height. The image width and height directory contains one subdirectory for each file format. The file format directory contains one or more files with the image width, image height, and file format specified by the name of its parent directories.

If the media directory is moved to a different location or a different media directory is used for testing, the symbol `media` in the Tcl test script can be changed to the location of the media directory, or the new media directory can be specified as a command-line argument.

Suppose that the name of a test case is `boxes`, this test case contains images with dimensions of width 1920 by height 1080 and width 1280 by height 720, each image size includes images with file formats `b64a` and `byr4`, and there is one image file for each combination of image dimensions and file format, then there are four pathnames relative to the media directory for image files:

```
$(ROOT)/media/boxes/1920x1080/b64a/boxes-1920x1080-0000.b64a

$(ROOT)/media/boxes/1920x1080/byr4/boxes-1920x1080-0000.byr4

$(ROOT)/media/boxes/1280x720/b64a/boxes-1280x720-0000.b64a

$(ROOT)/media/boxes/1280x720/byr4/boxes-1280x720-0000.byr4
```

The directory names in the pathname specify the image dimensions and file format. Each image filename includes the name of the test case, the image dimensions, the sequence number that distinguishes multiple images within a single directory, and the filename extension that indicates the file format.

The Tcl test script traverses the directory tree for each test case. Some of the image file formats do not contain the image dimensions or file format, so the directory name and filename extension are used to determine the dimensions and format of the image. For example, the pathname

```
$(ROOT)/media/boxes/1920x1080/b64a/boxes-1920x1080-0001.b64a
```

specifies that the image width is 1920, the image height is 1080, and the file format is `b64a` as specified by either the filename extension or the name of the directory that contains the image file.

The Tcl test script obtains the image dimensions from the directory name, for example by splitting the directory name `1920x1080` and the letter "`x`" into the width and height. The width and height are passed to the sample encoder and reference decoder programs as command-line arguments (see Table 2 and Table 3). The sample encoder and reference decoder programs use the filename extension to determine the file format. The Tcl test script can be modified to use the file format in the pathname and pass this format to the sample encoder or reference decoder programs as command-line arguments.

The Tcl test script performs the following actions:

1. Traverses the hierarchical directory structure for each test case.
2. Creates a directory for test results in the test case directory (see below).
3. Invokes the sample encoder on images found in each test case and places the encoded bitstream in the results directory.
4. Invokes the reference decoder on each encoded bitstream and places the decoded image in the results directory.
5. If the test case contains a directory called `master`, then the Tcl test script compares the decoded image with the reference image in the master directory using a simple binary file comparison.

The directory for test results is a subdirectory path starting in the folder that contains the test media. The subdirectory path is formed from the location of the media directory, the name of the Tcl test script, the date, the part number supplied as a command-line argument to the test script, and the build configuration. For example:

```
$(ROOT)/media/boxes/results/testcodec-2015-05-04/part4/release
```

The Tcl test script reports the test cases that produced decoded images that do not exactly match the reference image. The comparison distinguishes between debug and release builds, because a debug build can insert extra information into the bitstream to assist in debugging.

The Tcl test script accepts the command-line arguments listed in Table 9. For example, to run the boxes and gradient test cases in the media directory installed along with the test materials using the debug build configuration of the sample encoder and reference decoder built using the make file in the root directory of the installed software:

```
testcodec.tcl –m ../media –b debug –t boxes,gradient
```

The sample encoder and reference decoder can be used without the test script by providing the parameters on the command line. For example, to encode one of the images provided with the test materials and decode the resulting bitstream:

```
encoder –w 1280 –h 720 boxes-1280x720-0000.byr4 boxes-1280x720-0000-byr4.vc5

decoder boxes-1280x720-0000-byr4.vc5 boxes-1280x720-0000-byr4-vc5.byr4
```

These example commands mimic the file naming conventions described in this section.

Table 9 — Tcl test script command-line options.

Command Line Option	Description
`-r <root directory>`	Top-level directory for the installation of the VC-5 test materials. (The default depends on the platform)
`-m <media directory>`	`Top-level directory that contains the test cases. (The default is $(ROOT)/media)`
`-b <configuration>`	Release or debug build configuration. (The default build configuration is release)
`-c <compiler>`	Tool used to build the encoder and decoder. The only documented value for the compiler is `make` which is the default value.
`-t <test cases>`	List of test cases to run, separated by commas without spaces
`-d`	Output a DPX file to provide a displayable picture of the decoded image. The pathname is derived from the pathname for the decoded image. The default is to not output a displayable picture.
`-v`	Verbose output (default is to suppress verbose output)
`-e`	Even more version output (for debugging). The default is to suppress extra output for debugging.
`-p <part>`	VC-5 part passed to the codec as a command-line parameter.
`-s <sections>`	Comma-separated list of section numbers enabled for part 6 encoding (only applies if part 6 is enabled).
`-l <logfile>`	Specify a log file for recording the work done by the test script.
`-a`	Perform all tests in the comprehensive test suite (see section 8.4)

8.4 Comprehensive Test Suite

The Tcl script file `testcodec-suite.tcl` defines a set of tests called the comprehensive test suite. This set of tests is intended for performing a full set of regression tests on all codec capabilities. The regression tests can be run by executing the command:

```
testcodec.tcl -a
```

If the `-a` switch is provided on the command line, then the `-t`, `-p`, and `-s` switches are ignored.

9 Conformance Specification

9.1 Test Materials

The materials for testing conformance to the VC-5 essence standard shall comprise the following items:

1. The reference bitstream files listed in Annex A,
2. The release configuration of the VC-5 reference decoder built according to the instructions provided with the test materials distribution (section 10),
3. The reference decoded image files listed in Annex B,
4. The release configuration of the VC-5 sample encoder built according to the instructions provided with the test materials distribution (section 10), and
5. Sample image files as described in section 6.8.

9.2 Bitstream Conformance

Conformance tests shall use the VC-5 reference decoder to verify conformance of the bitstream to the specified parts that are enabled in the reference decoder.

A bitstream shall be deemed conformant to the specified parts if and only if all of the following conditions are satisfied:

1. The specified parts are enabled at compile-time when the reference decoder is built,
2. The specified parts are enabled at runtime when the reference decoder is run, and
3. The reference decoder completes the decoding process for the bitstream without reporting any warnings or errors.

9.3 Decoder Conformance

A decoder shall be deemed conformant with the specified parts if and only if all of the following steps are executed successfully:

1. Apply the decoder implementation to each of the reference bitstream files for the specified parts, configuring the decoder to output the image format corresponding to the bitstream,
2. Decode each reference bitstream file without reporting any errors or warnings,
3. Output an image file(s) that equals the image file(s) corresponding to the bitstream.

Two image files are equal if and only if the image files have the same length (in bytes) and the values of all bytes in each file are equal.

A decoder implementation could output the decoded image in a format that is different from the format of the image input to the encoder but that would require a different notion of image equality that is out of scope.

9.4 Metadata Conformance

9.4.1 Metadata Conformance Overview (Informative)

Since metadata is inserted into a VC-5 bitstream as metadata chunks without affecting the remainder of the bitstream, testing encoder conformance only involves verifying that the metadata was correctly inserted as metadata chunks in the bitstream.

Likewise, testing decoder conformance only involves verifying that the metadata extracted from a reference bitstream is equivalent (as defined in section 9.4.4) to the metadata extracted from the bitstream by the reference decoder.

9.4.2 Encoder Metadata Conformance

A conformance test for encoding embedded metadata as specified in SMPTE ST 2073-7 shall, for all metadata files listed in Table D.1, return success when the following procedure is applied to each metadata file:

1. The encoder implementation inserts the input metadata file into a VC-5 bitstream,
2. The dumper (section 6.7) extracts the metadata chunks from the encoded VC-5 bitstream into a single binary file,
3. The binary file of extracted metadata is compared to the corresponding encoded metadata file listed in Table D.3, and
4. The result is success if and only if the two files compared in step 3 are identical.

9.4.3 Decoder Metadata Conformance

A conformance test for decoding embedded metadata as specified in SMPTE ST 2073-7 shall, for all reference bitstreams listed in Table D.2, return success when the following procedure is applied to each reference bitstream:

1. The decoder implementation extracts all metadata from the reference bitstream into a single file in the XML format defined by SMPTE ST 2073-7:2022 Annex A,
2. The XML file of extracted metadata is compared to the corresponding decoded metadata file listed in Table D.4, and
3. The result is success if and only if the two files compared in step 2 are equivalent as specified in section 9.4.4.

A decoder implementation can use a different representation for metadata than the XML format specified in SMPTE ST 2073-7:2022 Annex A, in which case adapting the conformance procedure specified in this section is out of scope. The implementation could include a tool to convert the non-standard representation to the XML format defined in SMPTE ST 2073-7:2022 Annex A.

9.4.4 Decoded Metadata Equivalence

As specified in SMPTE ST 2073-7 section 10,

1. A decoder implementation shall process metadata tuples from all metadata chunks in the order in which the metadata tuples occur in the bitstream, and
2. If there are duplicate metadata tuples with the same metadata tag, then only the last metadata tuple instance in the bitstream shall be used.

The `compare.py` metadata tool can be used to test the equivalence of two metadata files in XML format (see section 6.7).

10 Installing and Building the Test Materials

10.1 Test Materials Installation

The test materials are distributed by SMPTE. Install the test materials into an empty directory. This directory will be the root directory of the test materials distribution, designated in this document by the symbol `ROOT`.

10.2 Test Materials Contents

The test materials distribution installed in the root directory includes the following folders:

`$(ROOT)/common`

Source code that is common to both the sample encoder and reference decoder.

`$(ROOT)/encoder`

Source code for the sample encoder.

`$(ROOT)/decoder`

Source code for the reference decoder.

`$(ROOT)/tables`

Codebook used by the sample encoder and reference decoder.

`$(ROOT)/external`

Software developed by third parties that is used by the test materials.

`$(ROOT)/converter`

Source code for a C language program for converting between image file formats.

`$(ROOT)/comparer`

Source code for a C language program that can be used to compare the decoded image with the image that was input to the encoder. The comparer program is not currently used by the test scripts.

`$(ROOT)/scripts`

Scripts written in Tcl for testing the sample encoder and reference decoder.

`$(ROOT)/media`

Sample images and reference bitstreams that are compliant with the VC-5 essence standard.

10.3 Build Scripts and Documentation

The root directory of the test materials distribution contains the following files:

$(ROOT)/notes/install.html

$(ROOT)/notes/install.pdf

Detailed instructions on installing, building, and running the software in the test materials distribution in HTML and PDF formats.

$(ROOT)/notes/release.html

$(ROOT)/notes/release.pdf

Release notes in HTML and PDF formats.

$(ROOT)/Makefile

Top-level make file for building the debug and release configurations of the sample encoder and reference decoder. The top-level make file can also create detailed documentation from the source code. Refer to the readme file for details.

10.4 Metadata Software

Tools for generating and managing metadata and the data files used to create the metadata test cases listed in Annex C are in the folder `$(ROOT)/metadata` in the test materials distribution (version 5.0 and later). See section 6.7 for descriptions of the tools.

10.5 Test Materials Media

The media folder is divided into test cases with one folder per test case named after the test case:

$(ROOT)/media/boxes

Sample images and reference bitstreams for the “boxes” test case.

$(ROOT)/media/gradient

Sample images and reference bitstreams for the “gradient” test case.

$(ROOT)/media/solid

Sample images and reference bitstreams for the “solid” test case.

$(ROOT)/media/sections

Sample images and reference bitstreams for testing image sections as defined in VC-5 Part 6. The image sections test case differs from the other test cases due to the unique requirements for testing image sections (see section 6.8.4).

Each test case is divided into sub-directories for the image dimensions. For example, all image files for the boxes test case with dimensions 1280 by 720 are in the folder `$(ROOT)/media/boxes/1280x720`.

The sub-directory of each image dimension is divided into sub-directories by pixel format. For example, all image files for the boxes test case with dimensions 1920 by 1080 in the Bayer pixel format are in the folder `$(ROOT)/media/boxes/1280x720/byr4`.

Encoded bitstreams corresponding to the test images and the results of decoding the bitstreams are in the master folder for each test case, subdivided by VC-5 part and build configuration. For example, the folder `$(ROOT)/media/boxes/master/part4/release` contains the bitstreams and decoded image files for the boxes test case executed with the release configuration of the encoder and decoder for VC-5 part 4.

`$(ROOT)/media/metadata`

Files of metadata as defined in SMPTE ST 2073-7 in the XML format specified in Annex A of that standard. Each file of metadata corresponds to a metadata test case as described in Annex C.

10.6 Building the Encoder and Decoder

The debug and release configurations of the sample encoder and reference decoder can be built by typing the command

```
make
```

in a terminal window open to the root directory of the test materials installation.

Other build options are described in the documents provided with the test materials distribution.

Annex A Reference Bitstreams (Normative)

A.1 Bitstream Files

The reference bitstreams included with the test materials are listed in the following sections by VC-5 part number. The bitstreams are compliant with the part of the VC-5 essence standard indicated by the section heading and the pathname. The bitstreams were encoded using a release build of the sample encoder. Each reference bitstream represents a single image.

The reference bitstreams were designed to test aspects of encoder and decoder implementations that are prone to errors such as the handling of images with width or height that is an odd number and images with smallest dimensions that are allowed by the VC-5 essence standard.

The reference images were chosen to detect errors in the wavelet transform implementation: images in the boxes test cases have hard edges, images in the gradient test case have a smooth color gradient, and images in the solid test case show wavelet transform errors at the image borders.

The reference bitstream files are located in the master sub-directory in each test case organized by VC-5 part number and build configuration. For example, the master bitstream files for the boxes test case, VC-5 part 1, release configuration, are in the directory:

```
$(ROOT)/media/boxes/master/part1/release
```

A.2 VC-5 Part 1 Bitstreams

```
boxes-1280x720-0000-part1-1280x720-byr4.vc5

boxes-1280x720-0000-part1-1280x720-rg48.vc5

boxes-1280x720-0001-part1-1280x720-byr4.vc5

boxes-1280x720-0001-part1-1280x720-rg48.vc5

boxes-1920x1080-0000-part1-1920x1080-byr4.vc5

boxes-1920x1080-0000-part1-1920x1080-rg48.vc5

boxes-1920x1080-0001-part1-1920x1080-byr4.vc5

boxes-1920x1080-0001-part1-1920x1080-rg48.vc5

boxes-640x480-0000-part1-640x480-byr4.vc5

boxes-640x480-0000-part1-640x480-rg48.vc5

boxes-640x480-0001-part1-640x480-byr4.vc5

boxes-640x480-0001-part1-640x480-rg48.vc5

gradient-1280x720-0000-part1-1280x720-byr4.vc5

gradient-1280x720-0001-part1-1280x720-byr4.vc5

gradient-640x480-0000-part1-640x480-b64a.vc5

gradient-640x480-0001-part1-640x480-b64a.vc5

solid-1280x720-0000-part1-1280x720-byr4.vc5
```


solid-1280x720-0001-part1-1280x720-byr4.vc5

solid-48x48-0000-part1-48x48-b64a.vc5

solid-48x48-0000-part1-48x48-rg48.vc5

solid-48x48-0001-part1-48x48-b64a.vc5

solid-48x48-0001-part1-48x48-rg48.vc5

solid-48x49-0000-part1-48x49-rg48.vc5

solid-48x49-0001-part1-48x49-rg48.vc5

solid-49x48-0000-part1-49x48-rg48.vc5

solid-49x48-0001-part1-49x48-rg48.vc5

solid-49x49-0000-part1-49x49-rg48.vc5

solid-49x49-0001-part1-49x49-rg48.vc5

solid-640x480-0000-part1-640x480-rg48.vc5

solid-640x480-0001-part1-640x480-rg48.vc5

solid-96x96-0000-part1-96x96-byr4.vc5

solid-96x96-0001-part1-96x96-byr4.vc5

solid-96x97-0000-part1-96x97-byr4.vc5

solid-96x97-0001-part1-96x97-byr4.vc5

solid-97x96-0000-part1-97x96-byr4.vc5

solid-97x96-0001-part1-97x96-byr4.vc5

solid-97x97-0000-part1-97x97-byr4.vc5

solid-97x97-0001-part1-97x97-byr4.vc5

A.3 VC-5 Part 3 Bitstreams

boxes-1280x720-0000-part3-1280x720-byr4.vc5

boxes-1280x720-0000-part3-1280x720-rg48.vc5

boxes-1280x720-0001-part3-1280x720-byr4.vc5

boxes-1280x720-0001-part3-1280x720-rg48.vc5

boxes-1920x1080-0000-part3-1920x1080-byr4.vc5

boxes-1920x1080-0000-part3-1920x1080-rg48.vc5

boxes-1920x1080-0001-part3-1920x1080-byr4.vc5

boxes-1920x1080-0001-part3-1920x1080-rg48.vc5

boxes-640x480-0000-part3-640x480-byr4.vc5

boxes-640x480-0000-part3-640x480-rg48.vc5

boxes-640x480-0001-part3-640x480-byr4.vc5

boxes-640x480-0001-part3-640x480-rg48.vc5

gradient-1280x720-0000-part3-1280x720-byr4.vc5

gradient-1280x720-0001-part3-1280x720-byr4.vc5

gradient-640x480-0000-part3-640x480-b64a.vc5

gradient-640x480-0001-part3-640x480-b64a.vc5

solid-1280x720-0000-part3-1280x720-byr4.vc5

solid-1280x720-0001-part3-1280x720-byr4.vc5

solid-48x48-0000-part3-48x48-b64a.vc5

solid-48x48-0000-part3-48x48-rg48.vc5

solid-48x48-0001-part3-48x48-b64a.vc5

solid-48x48-0001-part3-48x48-rg48.vc5

solid-48x49-0000-part3-48x49-rg48.vc5

solid-48x49-0001-part3-48x49-rg48.vc5

solid-49x48-0000-part3-49x48-rg48.vc5

solid-49x48-0001-part3-49x48-rg48.vc5

solid-49x49-0000-part3-49x49-rg48.vc5

solid-49x49-0001-part3-49x49-rg48.vc5

solid-640x480-0000-part3-640x480-rg48.vc5

solid-640x480-0001-part3-640x480-rg48.vc5

solid-96x96-0000-part3-96x96-byr4.vc5

solid-96x96-0001-part3-96x96-byr4.vc5

solid-96x97-0000-part3-96x97-byr4.vc5

solid-96x97-0001-part3-96x97-byr4.vc5

solid-97x96-0000-part3-97x96-byr4.vc5

solid-97x96-0001-part3-97x96-byr4.vc5

solid-97x97-0000-part3-97x97-byr4.vc5

solid-97x97-0001-part3-97x97-byr4.vc5

A.4 VC-5 Part 4 Bitstreams

boxes-1280x720-0000-part4-1280x720-nv12.vc5

boxes-1280x720-0001-part4-1280x720-nv12.vc5

boxes-1920x1080-0000-part4-1920x1080-nv12.vc5

boxes-1920x1080-0001-part4-1920x1080-nv12.vc5

boxes-640x480-0000-part4-640x480-nv12.vc5

boxes-640x480-0001-part4-640x480-nv12.vc5

boxes-96x96-0000-part4-96x96-nv12.vc5

boxes-96x96-0001-part4-96x96-nv12.vc5

gradient-1280x720-0000-part4-1280x720-nv12.vc5

gradient-1280x720-0001-part4-1280x720-nv12.vc5

gradient-96x96-0000-part4-96x96-nv12.vc5

gradient-96x96-0001-part4-96x96-nv12.vc5

solid-1280x720-0000-part4-1280x720-nv12.vc5

solid-1280x720-0001-part4-1280x720-nv12.vc5

solid-96x96-0000-part4-96x96-nv12.vc5

solid-96x96-0001-part4-96x96-nv12.vc5

A.5 VC-5 Part 5 Bitstreams

The VC-5 Part 5 bitstreams were encoded from all images with the same image dimensions and pixel format within each test case with VC-5 part 5 enabled.

boxes-part5-1280x720-byr4.vc5

boxes-part5-1280x720-rg48.vc5

boxes-part5-1920x1080-byr4.vc5

boxes-part5-1920x1080-rg48.vc5

boxes-part5-640x480-byr4.vc5

boxes-part5-640x480-rg48.vc5

gradient-part5-1280x720-byr4.vc5

gradient-part5-640x480-b64a.vc5

solid-part5-1280x720-byr4.vc5

solid-part5-48x48-b64a.vc5

solid-part5-48x48-rg48.vc5

solid-part5-48x49-rg48.vc5

solid-part5-49x48-rg48.vc5

solid-part5-49x49-rg48.vc5

solid-part5-640x480-rg48.vc5

solid-part5-96x96-byr4.vc5

solid-part5-96x97-byr4.vc5

solid-part5-97x96-byr4.vc5

solid-part5-97x97-byr4.vc5

A.6 VC-5 Part 6 Bitstreams

A.6.1 VC-5 Part 6 Test Cases

The VC-5 Part 6 bitstreams were encoded from the same images used to create the VC-5 Part 1 bitstreams, but with VC-5 Part 6 enabled and section numbers 2 through 9 enabled.

Two sets of reference bitstreams are provided:

1. Bitstreams created with all sections except the image section, enabled at runtime by specifying section types 2 through 6, inclusive, using the sections command-line argument (see Table 4).

2. Bitstreams created from two images with different dimensions using the image section type specified using the sections command-line argument (see Table 4).

A.6.2 Non-Image Section Bitstreams

Reference bitstreams encoded from the corresponding reference images as determined by the file naming conventions used in this recommended practice with all section types enabled except the image section type:

```
boxes-1280x720-0000-part6-1280x720-byr4.vc5

boxes-1280x720-0000-part6-1280x720-rg48.vc5

boxes-1280x720-0001-part6-1280x720-byr4.vc5

boxes-1280x720-0001-part6-1280x720-rg48.vc5

boxes-1920x1080-0000-part6-1920x1080-byr4.vc5

boxes-1920x1080-0000-part6-1920x1080-rg48.vc5

boxes-1920x1080-0001-part6-1920x1080-byr4.vc5

boxes-1920x1080-0001-part6-1920x1080-rg48.vc5

boxes-640x480-0000-part6-640x480-byr4.vc5

boxes-640x480-0000-part6-640x480-rg48.vc5

boxes-640x480-0001-part6-640x480-byr4.vc5

boxes-640x480-0001-part6-640x480-rg48.vc5

gradient-1280x720-0000-part6-1280x720-byr4.vc5

gradient-1280x720-0001-part6-1280x720-byr4.vc5

gradient-640x480-0000-part6-640x480-b64a.vc5

gradient-640x480-0001-part6-640x480-b64a.vc5

solid-1280x720-0000-part6-1280x720-byr4.vc5

solid-1280x720-0001-part6-1280x720-byr4.vc5

solid-48x48-0000-part6-48x48-b64a.vc5

solid-48x48-0000-part6-48x48-rg48.vc5

solid-48x48-0001-part6-48x48-b64a.vc5
```

```
solid-48x48-0001-part6-48x48-rg48.vc5

solid-48x49-0000-part6-48x49-rg48.vc5

solid-48x49-0001-part6-48x49-rg48.vc5

solid-49x48-0000-part6-49x48-rg48.vc5

solid-49x48-0001-part6-49x48-rg48.vc5

solid-49x49-0000-part6-49x49-rg48.vc5

solid-49x49-0001-part6-49x49-rg48.vc5

solid-640x480-0000-part6-640x480-rg48.vc5

solid-640x480-0001-part6-640x480-rg48.vc5

solid-96x96-0000-part6-96x96-byr4.vc5

solid-96x96-0001-part6-96x96-byr4.vc5

solid-96x97-0000-part6-96x97-byr4.vc5

solid-96x97-0001-part6-96x97-byr4.vc5

solid-97x96-0000-part6-97x96-byr4.vc5

solid-97x96-0001-part6-97x96-byr4.vc5

solid-97x97-0000-part6-97x97-byr4.vc5

solid-97x97-0001-part6-97x97-byr4.vc5
```

A.6.3 Image Section Bitstreams

Image sections require a different directory structure for organizing the test media since each image section can represent an image with different dimensions and pixel format.

The folder `$(ROOT)/media/sections` is subdivided into separate directories for each image sections test case: boxes and mixed. Each image sections test case contains the set of images that will be encoded into a single bitstream with each image in a separate image section in the bitstream creating one bitstream for each image sections test case. The bitstreams are located in the master folder within the image sections test case:

The two bitstreams for image sections testing included in the test materials are:

```
$(ROOT)/media/sections/master/boxes/release/boxes.vc5
$(ROOT)/media/sections/master/mixed/release/mixed.vc5
```

A.6.4 Image Section Layer Bitstreams

Image sections that contain layers require a different directory structure for organizing the test media.

The bitstream created by the sample encoder to test image sections with nested layers is as follows:

```
$(ROOT)/media/sections/layers/boxes-section-layers-rg48.vc5
```

Annex B Reference Images (Normative)

B.1 Image Files

The test materials include the image files that were decoded using a release build of the reference decoder from the reference bitstreams.

The decoded image files are located in the master sub-directory in each test case organized by VC-5 part number and build configuration. For example, the decoded image files for the boxes test case, VC-5 part 1, release configuration, are in the directory `$(ROOT)/media/boxes/master/part1/release`.

Decoded image files with subsampled color difference components (`nv12`) are only relevant for VC-5 Part 4.

B.2 VC-5 Part 1 Images

```
boxes-1280x720-0000-part1-1280x720-byr4-vc5.byr4

boxes-1280x720-0000-part1-1280x720-rg48-vc5.rg48

boxes-1280x720-0001-part1-1280x720-byr4-vc5.byr4

boxes-1280x720-0001-part1-1280x720-rg48-vc5.rg48

boxes-1920x1080-0000-part1-1920x1080-byr4-vc5.byr4

boxes-1920x1080-0000-part1-1920x1080-rg48-vc5.rg48

boxes-1920x1080-0001-part1-1920x1080-byr4-vc5.byr4

boxes-1920x1080-0001-part1-1920x1080-rg48-vc5.rg48

boxes-640x480-0000-part1-640x480-byr4-vc5.byr4

boxes-640x480-0000-part1-640x480-rg48-vc5.rg48

boxes-640x480-0001-part1-640x480-byr4-vc5.byr4

boxes-640x480-0001-part1-640x480-rg48-vc5.rg48

gradient-1280x720-0000-part1-1280x720-byr4-vc5.byr4

gradient-1280x720-0001-part1-1280x720-byr4-vc5.byr4

gradient-640x480-0000-part1-640x480-b64a-vc5.b64a

gradient-640x480-0001-part1-640x480-b64a-vc5.b64a

solid-1280x720-0000-part1-1280x720-byr4-vc5.byr4

solid-1280x720-0001-part1-1280x720-byr4-vc5.byr4

solid-48x48-0000-part1-48x48-b64a-vc5.b64a

solid-48x48-0000-part1-48x48-rg48-vc5.rg48

solid-48x48-0001-part1-48x48-b64a-vc5.b64a

solid-48x48-0001-part1-48x48-rg48-vc5.rg48
```


solid-48x49-0000-part1-48x49-rg48-vc5.rg48

solid-48x49-0001-part1-48x49-rg48-vc5.rg48

solid-49x48-0000-part1-49x48-rg48-vc5.rg48

solid-49x48-0001-part1-49x48-rg48-vc5.rg48

solid-49x49-0000-part1-49x49-rg48-vc5.rg48

solid-49x49-0001-part1-49x49-rg48-vc5.rg48

solid-640x480-0000-part1-640x480-rg48-vc5.rg48

solid-640x480-0001-part1-640x480-rg48-vc5.rg48

solid-96x96-0000-part1-96x96-byr4-vc5.byr4

solid-96x96-0001-part1-96x96-byr4-vc5.byr4

solid-96x97-0000-part1-96x97-byr4-vc5.byr4

solid-96x97-0001-part1-96x97-byr4-vc5.byr4

solid-97x96-0000-part1-97x96-byr4-vc5.byr4

solid-97x96-0001-part1-97x96-byr4-vc5.byr4

solid-97x97-0000-part1-97x97-byr4-vc5.byr4

solid-97x97-0001-part1-97x97-byr4-vc5.byr4

B.3 VC-5 Part 3 Images

boxes-1280x720-0000-part3-1280x720-byr4-vc5.byr4

boxes-1280x720-0000-part3-1280x720-rg48-vc5.rg48

boxes-1280x720-0001-part3-1280x720-byr4-vc5.byr4

boxes-1280x720-0001-part3-1280x720-rg48-vc5.rg48

boxes-1920x1080-0000-part3-1920x1080-byr4-vc5.byr4

boxes-1920x1080-0000-part3-1920x1080-rg48-vc5.rg48

boxes-1920x1080-0001-part3-1920x1080-byr4-vc5.byr4

boxes-1920x1080-0001-part3-1920x1080-rg48-vc5.rg48

boxes-640x480-0000-part3-640x480-byr4-vc5.byr4

boxes-640x480-0000-part3-640x480-rg48-vc5.rg48

boxes-640x480-0001-part3-640x480-byr4-vc5.byr4

boxes-640x480-0001-part3-640x480-rg48-vc5.rg48

gradient-1280x720-0000-part3-1280x720-byr4-vc5.byr4

gradient-1280x720-0001-part3-1280x720-byr4-vc5.byr4

gradient-640x480-0000-part3-640x480-b64a-vc5.b64a

gradient-640x480-0001-part3-640x480-b64a-vc5.b64a

solid-1280x720-0000-part3-1280x720-byr4-vc5.byr4

solid-1280x720-0001-part3-1280x720-byr4-vc5.byr4

solid-48x48-0000-part3-48x48-b64a-vc5.b64a

solid-48x48-0000-part3-48x48-rg48-vc5.rg48

solid-48x48-0001-part3-48x48-b64a-vc5.b64a

solid-48x48-0001-part3-48x48-rg48-vc5.rg48

solid-48x49-0000-part3-48x49-rg48-vc5.rg48

solid-48x49-0001-part3-48x49-rg48-vc5.rg48

solid-49x48-0000-part3-49x48-rg48-vc5.rg48

solid-49x48-0001-part3-49x48-rg48-vc5.rg48

solid-49x49-0000-part3-49x49-rg48-vc5.rg48

solid-49x49-0001-part3-49x49-rg48-vc5.rg48

solid-640x480-0000-part3-640x480-rg48-vc5.rg48

solid-640x480-0001-part3-640x480-rg48-vc5.rg48

solid-96x96-0000-part3-96x96-byr4-vc5.byr4

solid-96x96-0001-part3-96x96-byr4-vc5.byr4

solid-96x97-0000-part3-96x97-byr4-vc5.byr4

solid-96x97-0001-part3-96x97-byr4-vc5.byr4

solid-97x96-0000-part3-97x96-byr4-vc5.byr4

solid-97x96-0001-part3-97x96-byr4-vc5.byr4

solid-97x97-0000-part3-97x97-byr4-vc5.byr4

solid-97x97-0001-part3-97x97-byr4-vc5.byr4

B.4 VC-5 Part 4 Images

boxes-1280x720-0000-part4-1280x720-nv12-vc5.nv12

boxes-1280x720-0001-part4-1280x720-nv12-vc5.nv12

boxes-1920x1080-0000-part4-1920x1080-nv12-vc5.nv12

boxes-1920x1080-0001-part4-1920x1080-nv12-vc5.nv12

boxes-640x480-0000-part4-640x480-nv12-vc5.nv12

boxes-640x480-0001-part4-640x480-nv12-vc5.nv12

boxes-96x96-0000-part4-96x96-nv12-vc5.nv12

boxes-96x96-0001-part4-96x96-nv12-vc5.nv12

gradient-1280x720-0000-part4-1280x720-nv12-vc5.nv12

gradient-1280x720-0001-part4-1280x720-nv12-vc5.nv12

gradient-96x96-0000-part4-96x96-nv12-vc5.nv12

gradient-96x96-0001-part4-96x96-nv12-vc5.nv12

solid-1280x720-0000-part4-1280x720-nv12-vc5.nv12

solid-1280x720-0001-part4-1280x720-nv12-vc5.nv12

solid-96x96-0000-part4-96x96-nv12-vc5.nv12

solid-96x96-0001-part4-96x96-nv12-vc5.nv12

B.5 VC-5 Part 5 Images

A decoder compliant with VC-5 Part 5 will output both layers as separate images. A decoder that is not complaint with VC-5 Part 5 will output the image corresponding to the first layer encountered in the bitstream.

boxes-1280x720-0000-part5-1280x720-byr4-vc5.byr4

boxes-1280x720-0000-part5-1280x720-rg48-vc5.rg48

boxes-1280x720-0001-part5-1280x720-byr4-vc5.byr4

boxes-1280x720-0001-part5-1280x720-rg48-vc5.rg48

boxes-1920x1080-0000-part5-1920x1080-byr4-vc5.byr4

boxes-1920x1080-0000-part5-1920x1080-rg48-vc5.rg48

boxes-1920x1080-0001-part5-1920x1080-byr4-vc5.byr4

boxes-1920x1080-0001-part5-1920x1080-rg48-vc5.rg48

boxes-640x480-0000-part5-640x480-byr4-vc5.byr4

boxes-640x480-0000-part5-640x480-rg48-vc5.rg48

boxes-640x480-0001-part5-640x480-byr4-vc5.byr4

boxes-640x480-0001-part5-640x480-rg48-vc5.rg48

gradient-1280x720-0000-part5-1280x720-byr4-vc5.byr4

gradient-1280x720-0001-part5-1280x720-byr4-vc5.byr4

gradient-640x480-0000-part5-640x480-b64a-vc5.b64a

gradient-640x480-0001-part5-640x480-b64a-vc5.b64a

solid-1280x720-0000-part5-1280x720-byr4-vc5.byr4

solid-1280x720-0001-part5-1280x720-byr4-vc5.byr4

solid-48x48-0000-part5-48x48-b64a-vc5.b64a

solid-48x48-0000-part5-48x48-rg48-vc5.rg48

solid-48x48-0001-part5-48x48-b64a-vc5.b64a

solid-48x48-0001-part5-48x48-rg48-vc5.rg48

solid-48x49-0000-part5-48x49-rg48-vc5.rg48

solid-48x49-0001-part5-48x49-rg48-vc5.rg48

solid-49x48-0000-part5-49x48-rg48-vc5.rg48

solid-49x48-0001-part5-49x48-rg48-vc5.rg48

solid-49x49-0000-part5-49x49-rg48-vc5.rg48

solid-49x49-0001-part5-49x49-rg48-vc5.rg48

solid-640x480-0000-part5-640x480-rg48-vc5.rg48

solid-640x480-0001-part5-640x480-rg48-vc5.rg48

solid-96x96-0000-part5-96x96-byr4-vc5.byr4

solid-96x96-0001-part5-96x96-byr4-vc5.byr4

solid-96x97-0000-part5-96x97-byr4-vc5.byr4

solid-96x97-0001-part5-96x97-byr4-vc5.byr4

solid-97x96-0000-part5-97x96-byr4-vc5.byr4

solid-97x96-0001-part5-97x96-byr4-vc5.byr4

solid-97x97-0000-part5-97x97-byr4-vc5.byr4

solid-97x97-0001-part5-97x97-byr4-vc5.byr4

B.6 VC-5 Part 6 Images

B.6.1 Non-Image Section Images

The output images from a VC-5 Part 6 compliant decoder are not provided since the resulting images will be identical to the output of a decoder that is not compliant with VC-5 Part 6.

The VC-5 reference decoder creates a log file with the type and size of each section encountered in the bitstream, in bitstream order. The log files corresponding to the bitstreams listed in Annex A.6 are as listed as follows

boxes-1280x720-0000-part6-1280x720-byr4.log

boxes-1280x720-0000-part6-1280x720-rg48.log

boxes-1280x720-0001-part6-1280x720-byr4.log

boxes-1280x720-0001-part6-1280x720-rg48.log

boxes-1920x1080-0000-part6-1920x1080-byr4.log


```
boxes-1920x1080-0000-part6-1920x1080-rg48.log
boxes-1920x1080-0001-part6-1920x1080-byr4.log
boxes-1920x1080-0001-part6-1920x1080-rg48.log
boxes-640x480-0000-part6-640x480-byr4.log
boxes-640x480-0000-part6-640x480-rg48.log
boxes-640x480-0001-part6-640x480-byr4.log
boxes-640x480-0001-part6-640x480-rg48.log
gradient-1280x720-0000-part6-1280x720-byr4.log
gradient-1280x720-0001-part6-1280x720-byr4.log
gradient-640x480-0000-part6-640x480-b64a.log
gradient-640x480-0001-part6-640x480-b64a.log
solid-1280x720-0000-part6-1280x720-byr4.log
solid-1280x720-0001-part6-1280x720-byr4.log
solid-48x48-0000-part6-48x48-b64a.log
solid-48x48-0000-part6-48x48-rg48.log
solid-48x48-0001-part6-48x48-b64a.log
solid-48x48-0001-part6-48x48-rg48.log
solid-48x49-0000-part6-48x49-rg48.log
solid-48x49-0001-part6-48x49-rg48.log
solid-49x48-0000-part6-49x48-rg48.log
solid-49x48-0001-part6-49x48-rg48.log
solid-49x49-0000-part6-49x49-rg48.log
solid-49x49-0001-part6-49x49-rg48.log
solid-640x480-0000-part6-640x480-rg48.log
solid-640x480-0001-part6-640x480-rg48.log
solid-96x96-0000-part6-96x96-byr4.log
solid-96x96-0001-part6-96x96-byr4.log
solid-96x97-0000-part6-96x97-byr4.log
solid-96x97-0001-part6-96x97-byr4.log
solid-97x96-0000-part6-97x96-byr4.log
solid-97x96-0001-part6-97x96-byr4.log
```

```
solid-97x97-0000-part6-97x97-byr4.log

solid-97x97-0001-part6-97x97-byr4.log
```

B.6.2 Image Section Images

The decoded images for each image sections test case are as follows:

```
$(ROOT)/media/sections/boxes/boxes-1280x720-0000.rg48

$(ROOT)/media/sections/boxes/boxes-640x480-0000.rg48

$(ROOT)/media/sections/boxes/boxes-640x480-0001.rg48

$(ROOT)/media/sections/mixed/boxes-1920x1080-0000.byr4

$(ROOT)/media/sections/mixed/boxes-1920x1080-0000.rg48

$(ROOT)/media/sections/mixed/boxes-1920x1080-0001.byr4

$(ROOT)/media/sections/mixed/boxes-1920x1080-0001.rg48
```

B.6.3 Image Section Layer Images

The decoded images for the image section layers test case are as follows:

```
boxes-1280x720-0000-rg48-vc5.rg48

boxes-640x480-0000-rg48-vc5.rg48

boxes-640x480-0001-rg48-vc5.rg48
```

Annex C Reference Metadata (Informative)

C.1 Metadata Test Cases

Test cases are included for all categories of metadata defined in SMPTE ST 2073-7 as specified in the following sections.

Annexes C.2 through C.6 provide test data for each of the metadata classes defined in SMPTE ST 2073-7:2022, section 8.2 and Annexes B through J.

SMPTE ST 2073-7:2022 defines the representation of metadata in a VC-5 bitstream (see sections 6 and 7 and Table 4) and a plain-text XML representation for metadata (Annex A). Blocks of arbitrary sized binary data (data type code 'B') are represented in the bitstream as a string of bytes. The software described in this recommended practice uses base64 as the plain-text representation for blocks of binary data. The base64 format is described in IETF RFC 4648.

In the JSON and XML representations of metadata, binary metadata values (data type code 'B') are encoded in base64.

The JSON file format is described in IETF RFC 8259 and the XML file format is described in W3C Recommendation Extensible Markup Language (XML).

C.2 Intrinsic Metadata

C.2.1 Intrinsic Metadata Overview

Intrinsic metadata as defined in SMPTE ST 2073-7:2022 Annex B describe characteristics of the image represented by the bitstream that are not fully described by the tag-value pairs that determine how the image is decoded. For example, the image can be decoded without knowing the color space of the original image, but a decoder implementation can use that metadata in the image repacking process to convert the decoded image to the color space required by the application.

Metadata is inserted into the bitstream in one or more chunks (SMPTE ST 2073-7:2022, section 9). Each metadata chunk contains one or more metadata tuples and each metadata tuple represents one item of metadata. If multiple instances of a metadata item are present in the bitstream, only the last item in the bitstream is decoded, the earlier items are ignored (SMPTE ST 2073-7:2022, section 10).

For the purposes of creating test cases for intrinsic metadata, the workflow is designed around three categories of intrinsic metadata:

1. Simple metadata test cases comprising metadata tuples that do not contain other tuples,
2. Nested metadata test cases with one or more metadata tuples in the text of each nested metadata tuple,
3. Complex metadata test cases comprising a combination of simple and nested metadata tuples.

C.2.2 Simple Intrinsic Metadata

The simple intrinsic metadata test cases are listed in Table C.1. The filename suffix indicates whether the test cases contain metadata tuples that have been randomized (character "r"), include duplicate metadata items (character "d"), and the number of metadata chunks (character "c" followed by digits for the number of chunks). For example, the suffix "rdc3" would indicate that the testcase contains randomized metadata tuples with duplicates partitioned into three metadata chunks.

The simple intrinsic metadata test cases with different chunk sizes are intended to test the ability of a VC-5 codec implementation to handle tuple split across multiple chunks. The simple intrinsic metadata test cases with duplicates are intended to test the ability of a VC-5 codec implementation to handle duplicate tuples with the same metadata tag.

The XML files for the simple intrinsic metadata test cases listed in Table C.1 are in the test materials directory:

```
$(ROOT)/media/metadata/intrinsic/simple
```

Table C.1 — Simple intrinsic metadata test cases.

Filename	Description
simple-c01.xml	All simple intrinsic metadata tuples in 1 chunk without duplicates
simple-c02.xml	All simple intrinsic metadata tuples in 2 chunks without duplicates
simple-c03.xml	All simple intrinsic metadata tuples in 3 chunks without duplicates
simple-c04.xml	All simple intrinsic metadata tuples in 4 chunks without duplicates
simple-c05.xml	All simple intrinsic metadata tuples in 5 chunks without duplicates
simple-c06.xml	All simple intrinsic metadata tuples in 6 chunks without duplicates
simple-c07.xml	All simple intrinsic metadata tuples in 7 chunks without duplicates
simple-c08.xml	All simple intrinsic metadata tuples in 8 chunks without duplicates
simple-c09.xml	All simple intrinsic metadata tuples in 9 chunks without duplicates
simple-c10.xml	All simple intrinsic metadata tuples in 10 chunks without duplicates
simple-c11.xml	All simple intrinsic metadata tuples in 11 chunks without duplicates
simple-c12.xml	All simple intrinsic metadata tuples in 12 chunks without duplicates
simple-c13.xml	All simple intrinsic metadata tuples in 13 chunks without duplicates
simple-c14.xml	All simple intrinsic metadata tuples in 14 chunks without duplicates
simple-c15.xml	All simple intrinsic metadata tuples in 15 chunks without duplicates
simple-c16.xml	All simple intrinsic metadata tuples in 16 chunks without duplicates
simple-c17.xml	All simple intrinsic metadata tuples in 17 chunks without duplicates
simple-c18.xml	All simple intrinsic metadata tuples in 18 chunks without duplicates
simple-c19.xml	All simple intrinsic metadata tuples in 19 chunks without duplicates
simple-c20.xml	All simple intrinsic metadata tuples in 20 chunks without duplicates
simple-c21.xml	All simple intrinsic metadata tuples in 21 chunks without duplicates
simple-c22.xml	All simple intrinsic metadata tuples in 22 chunks without duplicates
simple-c23.xml	All simple intrinsic metadata tuples in 23 chunks without duplicates
simple-c24.xml	All simple intrinsic metadata tuples in 24 chunks without duplicates
simple-c25.xml	All simple intrinsic metadata tuples in 25 chunks without duplicates

Filename	Description
simple-rdc1.xml	All simple intrinsic metadata tuples randomized into 1 chunk with duplicates that contain different values
simple-rdc2.xml	All simple intrinsic metadata tuples randomized into 2 chunks with duplicates that contain different values
simple-rdc4.xml	All simple intrinsic metadata tuples randomized into 4 chunks with duplicates that contain different values
simple-rdc7.xml	All simple intrinsic metadata tuples randomized into 7 chunks with duplicates that contain different values

C.2.3 Nested Intrinsic Metadata

The nested intrinsic metadata test cases are listed in Table C.2. The nested intrinsic metadata test cases are used to create the complex intrinsic metadata test cases described in Annex C.2.4 by combining combinations of simple and nested intrinsic metadata test cases.

The XML files for the nested intrinsic metadata test cases listed in Table C.2 are in the test materials directory:

```
$(ROOT)/media/metadata/intrinsic/nested
```

Table C.2 — Nested intrinsic metadata test cases.

Filename	Description
encoding-LOGA.xml	Nested intrinsic metadata for the log encoding curve (SMPTE ST 2073-7:2022 section B.9.2)
encoding-GAMA.xml	Nested intrinsic metadata for the gamma encoding curve (SMPTE ST 2073-7:2022 section B.9.3)
encoding-LINR.xml	Nested intrinsic metadata for the linear encoding curve (SMPTE ST 2907-7:2022 section B.9.4)
encoding-FSLG.xml	Nested intrinsic metadata for the FS-Log encoding curve (SMPTE ST 2073-7:2022 section B.9.5)
encoding-LOGC.xml	Nested intrinsic metadata for the Log C encoding curve (SMPTE ST 2073-7:2022 section B.9.6)
encoding-PQEC.xml	Nested intrinsic metadata for the perceptual quantizer encoding curve (SMPTE ST 2073-7:2022 section B.9.7)
encoding-HLGE.xml	Nested intrinsic metadata for the hybrid log-gamma encoding curve (SMPTE ST 2073-7:2022 section B.9.8)
layers-stereo.xml	Nested intrinsic metadata for two layers representing the left and right images of a stereo pair (SMPTE ST 2073-7:2022 section B.13)
layers-interlaced.xml	Nested intrinsic metadata for the top and bottom fields of an interlaced image (SMPTE ST 2073-7:2022 section B.13)
layers-hdr.xml	Nested intrinsic metadata for three exposures of an HDR image (SMPTE ST 2073-7:2022 section B.13)

C.2.4 Complex Intrinsic Metadata

The complex intrinsic metadata test cases are listed in Table C.3. The complex intrinsic metadata test cases are formed by computing all combinations of the test cases in three categories of test cases:

1. The simple metadata test case `simple-c01.xml` from Table C.1,
2. The seven encoding curve metadata test cases listed in Table C.2, and
3. The three layers metadata test cases listed in Table C.2.

For example, the metadata test case `simple-LOGA-stereo.xml` is the combination all metadata tuples used to create the test cases listed in Table C.1 plus the LOGA encoding curve and stereo layer metadata listed in Table C.2.

The XML files for the complex intrinsic metadata test cases listed in Table C.3 are in the test materials directory:

```
$(ROOT)/media/metadata/intrinsic/complex
```

Table C.3 — Complex intrinsic metadata test cases.

Filename	Description
`simple-LOGA-stereo.xml`	Simple intrinsic metadata with log encoding curve and stereo layers metadata
`simple-LOGA-interlaced.xml`	Simple intrinsic metadata with log encoding curve and interlaced layers metadata
`simple-LOGA-hdr.xml`	Simple intrinsic metadata with log encoding curve and HDR layers metadata
`simple-GAMA-stereo.xml`	Simple intrinsic metadata with gamma encoding curve and stereo layers metadata
`simple-GAMA-interlaced.xml`	Simple intrinsic metadata with gamma encoding curve and interlaced layers metadata
`simple-GAMA-hdr.xml`	Simple intrinsic metadata with gamma encoding curve and HDR layers metadata
`simple-LINR-stereo.xml`	Simple intrinsic metadata with linear encoding curve and stereo layers metadata
`simple-LINR-interlaced.xml`	Simple intrinsic metadata with linear encoding curve and interlaced layers metadata
`simple-LINR-hdr.xml`	Simple intrinsic metadata with linear encoding curve and HDR layers metadata
`simple-FSLG-stereo.xml`	Simple intrinsic metadata with FS-Log encoding curve and stereo layers metadata
`simple-FSLG-interlaced.xml`	Simple intrinsic metadata with FS-Log encoding curve and interlaced layers metadata
`simple-FSLG-hdr.xml`	Simple intrinsic metadata with FS-Log encoding curve and HDR layers metadata

Filename	Description
simple-LOGC-stereo.xml	Simple intrinsic metadata with Log C encoding curve and stereo layers metadata
simple-LOGC-interlaced.xml	Simple intrinsic metadata with Log C encoding curve and interlaced layers metadata
simple-LOGC-hdr.xml	Simple intrinsic metadata with Log C encoding curve and HDR layers metadata
simple-PQEC-stereo.xml	Simple intrinsic metadata with perceptual quantizer encoding curve and stereo layers metadata
simple-PQEC-interlaced.xml	Simple intrinsic metadata with perceptual quantizer encoding curve and interlaced layers metadata
simple-PQEC-hdr.xml	Simple intrinsic metadata with perceptual quantizer encoding curve and HDR layers metadata
simple-HLGE-stereo.xml	Simple intrinsic metadata with hybrid log-gamma encoding curve and stereo layers metadata
simple-HLGE-interlaced.xml	Simple intrinsic metadata with hybrid log-gamma encoding curve and interlaced layers metadata
simple-HLGE-hdr.xml	Simple intrinsic metadata with hybrid log-gamma encoding curve and HDR layers metadata

C.3 Streaming Data

Streaming data enables embedding time series data into a VC-5 bitstream. Examples of streaming data include the camera model, unique ID, and GPS location.

The streaming data testcases are listed in Table C.4. The test data was obtained from the MP4 files in the `samples` folder in the GPMF Parser source-code distribution (URL provided in the bibliography). The `gpmf-parser` was modified to output streaming data in CSV format for use by the `generate.py` script (see section 6.7). The CSV file format is described in IETF RFC 4180.

The XML files for the testcases listed in Table C.4 are in the test materials directory:

```
$(ROOT)/media/metadata/streaming
```

Table C.4 — Streaming data test cases.

Filename	Description
hero5.xml	Streaming data extracted from the file hero5.mp4
hero6.xml	Streaming data extracted from the file hero6.mp4
hero6+ble.xml	Streaming data extracted from the file hero6+ble.mp4
hero7.xml	Streaming data extracted from the file hero7.mp4
hero8.xml	Streaming data extracted from the file hero8.mp4
karma.xml	Streaming data extracted from the file karma.mp4
Fusion.xml	Streaming data extracted from the file Fusion.mp4

Filename	Description
max-360mode.xml	Streaming data extracted from the file max-360mode.mp4
max-heromode.xml	Streaming data extracted from the file max-heromode.mp4

C.4 Extrinsic Metadata

C.4.1 Extrinsic Metadata Overview

Extrinsic metadata as defined in SMPTE ST 2073-7:2022 Annex E embeds in the VC-5 bitstream metadata that is defined in standards that are not published as part of the VC-5 standards suite.

C.4.2 XMP Metadata

SMPTE ST 2073-7:2022, Annex E, defines a mechanism for embedding XMP metadata in the VC-5 bitstream. Documentation, source code, and sample files are provided in the XMP Toolkit SDK listed in the bibliography. The extrinsic metadata testcases for XMP are listed in Table C.5. The XMP data embedded in the VC-5 extrinsic metadata instances (metadata tag 'XMPD') was extracted from JPEG files in the folder `samples/testfiles` distributed with the XMP Toolkit SDK.

The files of XMP metadata extracted from the JPEG files are located in the following directory:

```
$(ROOT)/metadata/media/xmp
```

The XML files for the testcases listed in Table C.5 are located in the following directory:

```
$(ROOT)/media/metadata/input
```

Table C.5 — XMP metadata test cases.

Filename	Description
Image1.xml	XMP metadata extracted from the file Image1.jpg using ExifTool
Image2.xml	XMP metadata extracted from the file Image2.jpg using ExifTool
BlueSquare.xml	XMP metadata extracted from the file BlueSquare.jpg using ExifTool

A URL for the ExifTool is provided in the bibliography.

C.4.3 DPX Metadata

SMPTE ST 2073-7:2022, Annex F, defines a mechanism for embedding a DPX file in the VC-5 bitstream. The DPX file header is described in SMPTE ST 268-1 and SMPTE ST 268-2. The extrinsic metadata testcases for DPX are listed in Table C.6.

The DPX file headers embedded in the VC-5 extrinsic metadata instances (metadata tag 'DPXF') were extracted from DPX files included in the folder `examples` distributed with the informative reference implementation for SMPTE ST 268-2 (the URL to the code repository on GitHub is provided in the bibliography).

The XML files for the testcases listed in Table C.6 are located in the following directory:

```
$(ROOT)/media/metadata/input
```

The file header contained in each XML file was extracted from the corresponding DPX file using the dpxdump tool (section 6.7).

Table C.6 — DPX file header metadata test cases.

Filename
hintergrund-musik.xml
ctest_10bpcfr_BGRA444p_lsbf_l2r_packed_rle.xml
ctest_10bpcfr_RGB444_lsbf_l2r_mb_norle.xml
ctest_10bpcfr_RGB444_msbf_l2r_ma_norle.xml
ctest_10bpcfr_RGBA444p_msbf_l2r_mb_rle.xml
ctest_10bpclr_ARGB444_lsbf_l2r_ma_norle.xml
ctest_10bpclr_ARGB444_msbf_l2r_packed_norle.xml
ctest_10bpclr_ARGB444p_lsbf_l2r_ma_rle.xml
ctest_10bpclr_ARGB444p_msbf_l2r_packed_rle.xml
ctest_10bpclr_BGRA444_lsbf_l2r_packed_norle.xml
ctest_10bpclr_CYAYA420_msbf_r2l_mb_rle.xml
ctest_10bpclr_CYY420_msbf_r2l_mb_norle.xml
ctest_10bpclr_CbYACrYA422_lsbf_r2l_ma_rle.xml
ctest_10bpclr_CbYACrYA422_msbf_r2l_packed_rle.xml
ctest_10bpclr_CbYCr444_lsbf_r2l_packed_norle.xml
ctest_10bpclr_CbYCrA444_lsbf_r2l_packed_rle.xml
ctest_10bpclr_CbYCrY422_lsbf_r2l_ma_norle.xml
ctest_10bpclr_CbYCrY422_msbf_r2l_packed_norle.xml
ctest_10bpclr_RGB444p_lsbf_l2r_mb_rle.xml
ctest_10bpclr_RGB444p_msbf_l2r_ma_rle.xml
ctest_10bpclr_RGBA444_msbf_l2r_mb_norle.xml
ctest_10bpclr_YCbCr422p_lsbf_r2l_mb_norle.xml
ctest_10bpclr_YCbCr422p_msbf_r2l_ma_norle.xml
ctest_10bpclr_YCbCrA422p_lsbf_r2l_mb_rle.xml
ctest_10bpclr_YCbCrA422p_msbf_r2l_ma_rle.xml
ctest_12bpcfr_BGR444_lsbf_r2l_mb_norle.xml
ctest_12bpcfr_BGR444_msbf_r2l_ma_norle.xml
ctest_12bpcfr_BGRA444p_msbf_r2l_mb_rle.xml

Filename
ctest_12bpcfr_RGBA444p_lsbf_l2r_packed_rle.xml
ctest_12bpclr_ABGR444_lsbf_l2r_ma_norle.xml
ctest_12bpclr_ABGR444_msbf_l2r_packed_norle.xml
ctest_12bpclr_ABGR444p_lsbf_l2r_ma_rle.xml
ctest_12bpclr_ABGR444p_msbf_l2r_packed_rle.xml
ctest_12bpclr_BGR444p_lsbf_r2l_mb_rle.xml
ctest_12bpclr_BGR444p_msbf_r2l_ma_rle.xml
ctest_12bpclr_BGRA444_msbf_r2l_mb_norle.xml
ctest_12bpclr_CYAYA420_lsbf_r2l_packed_rle.xml
ctest_12bpclr_CYY420_lsbf_r2l_packed_norle.xml
ctest_12bpclr_CbYACrYA422_msbf_l2r_mb_rle.xml
ctest_12bpclr_CbYCr444_lsbf_l2r_mb_norle.xml
ctest_12bpclr_CbYCr444_msbf_l2r_ma_norle.xml
ctest_12bpclr_CbYCrA444_lsbf_l2r_mb_rle.xml
ctest_12bpclr_CbYCrA444_msbf_l2r_ma_rle.xml
ctest_12bpclr_CbYCrY422_msbf_l2r_mb_norle.xml
ctest_12bpclr_RGBA444_lsbf_l2r_packed_norle.xml
ctest_12bpclr_YCbCr420p_lsbf_r2l_ma_norle.xml
ctest_12bpclr_YCbCr420p_msbf_r2l_packed_norle.xml
ctest_12bpclr_YCbCrA420p_lsbf_r2l_ma_rle.xml
ctest_12bpclr_YCbCrA420p_msbf_r2l_packed_rle.xml

C.4.4 MXF Metadata

SMPTE ST 2073-7:2022, Annex G, defines a mechanism for embedding MXF Annex F and G essence descriptors in the VC-5 bitstream. The MXF essence descriptors are described in ST 377-1. The extrinsic metadata testcases for MXF are listed in Table C.7.

The MXF Annex F and G essence descriptors embedded in the VC-5 extrinsic metadata instances (metadata tag ‘MXFD’) were extracted from MXF files are located in the following directory:

```
$(ROOT)/metadata/media/mxf
```

The essence descriptors were extracted from the MXF files using a modified version of the mxfdump tool (`https://sourceforge.net/projects/mxflib/files/mxflib/1.0.1/`).

The XML files for the testcases listed in Table C.7 are in the directory:

```
$(ROOT)/media/metadata/input
```

Table C.7 — MXF Annex F and G metadata test cases.

Filename	Description
part15-j2c.xml	Testcase for the MXF RGBA essence descriptor
yuv422_10b_p15.xml	Testcase for the MXF CDCI essence descriptor

C.4.5 ACES Metadata

SMPTE ST 2073-7:2022, Annex H, defines a mechanism for embedding ACES attributes in the VC-5 bitstream. ACES metadata is described in SMPTE ST 2065-4. The extrinsic metadata testcases for ACES are listed in Table C.8.

The ACES attributes embedded in the VC-5 extrinsic metadata instances (metadata tag 'ACES') were extracted from EXR files in the ACES folder within the images provided by reference in the AMPAS Academy Color Encoding System Developer Resources:

```
https://github.com/ampas/aces-dev/tree/master/images
```

Information about ACES can be obtained from the URL provided in the bibliography.

The EXR files are located in the following directory:

```
$(ROOT)/metadata/media/aces
```

The ACES attributes were extracted from the OpenEXR files using the exrdump tool (section 6.7). The URL to the OpenEXR web site is provided in the bibliography.

The XML files for the testcases listed in Table C.8 are located in the following directory:

```
$(ROOT)/media/metadata/input
```

Table C.8 — ACES metadata test cases.

Filename	Description
DigitalLAD.2048x1556.xml	ACES attributes extracted from the file DigitalLAD.2048x1556.exr
SonyF35.StillLife.xml	ACES attributes extracted from the file SonyF35.StillLife.exr
syntheticChart.01.xml	ACES attributes extracted from the file syntheticChart.01.exr

C.4.6 ALE Metadata

ALE is a plain text file format for the exchange of metadata in post-processing. Each row corresponds to a metadata record. In each row, columns are delineated by tab characters or commas. Each column corresponds to a type of metadata. Column headings identify the type of metadata. Tab separated values (TSV) is an IANA registered MIME data type: text/tab-separated-values. Comma separated values are defined in IETF RFC 4180.

SMPTE ST 2073-7:2022, Annex I, defines a mechanism for embedding ALE metadata in the VC-5 bitstream. The extrinsic metadata testcases for ALE are listed in Table C.9.

The ALE metadata embedded in the VC-5 extrinsic metadata instances (metadata tag ‘ALEM’) were exported from a video sequence comprising multiple clips from a single video file. Each line in the ALE metadata corresponds to a single video clip in the sequence.

The ALE files are located in the following directory:

```
$(ROOT)/metadata/media/aces
```

The XML files for the testcases listed in Table C.9 are located in the following directory:

```
$(ROOT)/media/metadata/input
```

Table C.9 — ALE metadata test cases.

Filename	Description
`aletest-seq1.xml`	Used non-linear editing software to create a sequence of sub-clips and exported the timeline to an ALE file

C.4.7 Dynamic Metadata for Color Volume Transform

SMPTE ST 2073-7:2022, Annex J, defines a mechanism for embedding dynamic metadata for color volume transform (DMCVT) in the VC-5 bitstream. Embedding of DMCVT metadata in an MXF file is described in SMPTE ST 2094-2. The extrinsic metadata testcases for DMCVT are listed in Table C.10.

The MXF files from which dynamic metadata for color volume transform (DMCVT) was extracted are located in the following directory:

```
$(ROOT)/metadata/media/dmcvt
```

Dynamic metadata for the color volume transform embedded in the VC-5 extrinsic metadata instances (metadata tag ‘DMCT’) was extracted using the following procedure:

1. Use a hex editor to extract the KLV tuple for the dynamic color volume transform from an MXF file into a file in bin hex format.
2. Use the `bintool` program to convert the file in bin hex format into a file in base64 format.
3. Use the `dmcvt.py` script to convert the metadata file in base64 format to a file in JSON format.
4. Use the `generate.py` script to convert the metadata file in JSON format into a metadata test case in XML format as specified in SMPTE ST 2073-7:2022 Annex A.

The tools used in the procedure listed above are described in section 6.7.

The XML files for the testcases listed in Table C.10 are located in the following directory:

```
$(ROOT)/media/metadata/input
```

Table C.10 — Dynamic metadata for color volume transform (DMCVT) test cases.

Filename	Description
`imf_as02_DMCVT.xml`	KLV tuple for the DMCVT metadata extracted from the file imf_as02_DMCVT.mxf

C.5 Dark Metadata

Dark metadata as defined in SMPTE ST 2073-7:2022 Annex D embeds in the VC-5 bitstream metadata that is not defined in any published standard. Examples of dark metadata include vendor-specific metadata.

The dark metadata described in this section was created by encoding a string of information about a fake vendor and fake camera into base64, essentially obscuring the metadata.

The dark metadata testcases are listed in Table C.11.

The XML files for the testcases listed in Table C.11 are located in the following directory:

```
$(ROOT)/media/metadata/input
```

Table C.11 — Dark metadata test cases.

Filename	Description
dark-fourcc.xml	Dark metadata containing a block of bytes in base64 format with a FOURCC for the identification code (metadata tag 'VENI')
dark-uuid.xml	Dark metadata containing a block of bytes in base64 format with a UUID for the identification code (metadata tag 'VENI')

C.6 Multiclass Metadata

In addition to the metadata test cases for individual metadata classes, metadata test cases have been generated that include metadata class instances from more than one metadata class.

The multiclass metadata test cases were created by combining the JSON representation of metadata test cases from multiple metadata classes into a single metadata test case in XML format using the `generate.py` script described in section 6.7.

A multiclass metadata test case can include more than one instance of a metadata class.

The XML files for the multiclass metadata testcases are located in the following directory:

```
$(ROOT)/media/metadata/input
```

Table C.12 lists the filename for the combination metadata test cases in XML format and the metadata test cases from the metadata classes that were incorporated into the combination metadata test case.

Table C.12 — Multiclass metadata test cases.

Filename Test Case Instances	Metadata	Description
multiclass-01.xml	simple-LOGA-stereo.xml BlueSquare.xml part15-j2c.xml syntheticChart.01.xml	Four metadata class instances from different metadata classes in a single metadata chunk

Filename Test Case Instances	Metadata	Description
multiclass-02.xml	simple-LOGA-stereo.xml part15-j2c.xml yuv422_10b_p15.xml	Three metadata class instances with two duplicate instances from the same metadata class (MXF extrinsic) in a single metadata chunk

Annex D Conformance Testing (Normative)

D.1 Conformance Testing Overview

This annex specifies the metadata files in XML format and the encoded metadata in binary format used for testing compliance of an encoder or decoder implementation with SMPTE ST 2073-7 as follows:

- Annex D.2 lists the metadata input to the sample encoder in XML format,
- Annex D.3 lists the encoded bitstreams that contain embedded metadata,
- Annex D.4 lists the metadata in binary format extracted from the encoded bitstream by the metadata chunk element dumper (section 6.7),
- Annex D.5 lists metadata decoded by the reference decoder in XML format.

All metadata in XML format conforms to the specification in SMPTE ST 2073-7:2022 Annex A.

Binary data can be in a file of binary bytes or encoded as base64.

D.2 Input Metadata

The metadata files in XML format input to the sample encoder for conformance testing as specified in section 9.4 are listed in Table D.1. Each metadata file corresponds by filename to a metadata test case described in Annexes C.2 through C.6.

The input metadata are in the test materials directory:

```
$(ROOT)/media/metadata/input
```

Table D.1 — Input metadata in XML format for conformance testing.

Filename	Description
simple-c01.xml	C.2.2 Simple Intrinsic Metadata
simple-rdc1.xml	C.2.2 Simple Intrinsic Metadata
simple-rdc2.xml	C.2.2 Simple Intrinsic Metadata
simple-rdc4.xml	C.2.2 Simple Intrinsic Metadata
simple-rdc7.xml	C.2.2 Simple Intrinsic Metadata
simple-LOGA-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-LOGA-interlaced.xml	C.2.4 Complex Intrinsic Metadata
simple-LOGA-hdr.xml	C.2.4 Complex Intrinsic Metadata
simple-GAMA-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-GAMA-interlaced.xml	C.2.4 Complex Intrinsic Metadata

Filename	Description
simple-GAMA-hdr.xml	C.2.4 Complex Intrinsic Metadata
simple-LINR-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-LINR-interlaced.xml	C.2.4 Complex Intrinsic Metadata
simple-LINR-hdr.xml	C.2.4 Complex Intrinsic Metadata
simple-FSLG-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-FSLG-interlaced.xml	C.2.4 Complex Intrinsic Metadata
simple-FSLG-hdr.xml	C.2.4 Complex Intrinsic Metadata
simple-LOGC-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-LOGC-interlaced.xml	C.2.4 Complex Intrinsic Metadata
simple-LOGC-hdr.xml	C.2.4 Complex Intrinsic Metadata
simple-PQEC-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-PQEC-interlaced.xml	C.2.4 Complex Intrinsic Metadata
simple-PQEC-hdr.xml	C.2.4 Complex Intrinsic Metadata
simple-HLGE-stereo.xml	C.2.4 Complex Intrinsic Metadata
simple-HLGE-interlaced.xml	C.2.4 Complex Intrinsic Metadata
simple-HLGE-hdr.xml	C.2.4 Complex Intrinsic Metadata
hero5.xml	C.3 Streaming Data
hero6.xml	C.3 Streaming Data
hero7.xml	C.3 Streaming Data
hero8.xml	C.3 Streaming Data
karma.xml	C.3 Streaming Data
hero6+ble.xml	C.3 Streaming Data
fusion.xml	C.3 Streaming Data

Filename	Description
max-360mode.xml	C.3 Streaming Data
max-heromode.xml	C.3 Streaming Data
Image1.xml	C.4.2 XMP Metadata
Image2.xml	C.4.2 XMP Metadata
BlueSquare.xml	C.4.2 XMP Metadata
ctest_10bpcfr_BGRA444p_lsbf_l2r_packed_rle.xml	C.4.3 DPX Metadata
ctest_10bpcfr_RGB444_lsbf_l2r_mb_norle.xml	C.4.3 DPX Metadata
ctest_10bpcfr_RGB444_msbf_l2r_ma_norle.xml	C.4.3 DPX Metadata
ctest_10bpcfr_RGBA444p_msbf_l2r_mb_rle.xml	C.4.3 DPX Metadata
ctest_10bpclr_CbYCr444_lsbf_r2l_packed_norle.xml	C.4.3 DPX Metadata
ctest_10bpclr_CbYCrA444_lsbf_r2l_packed_rle.xml	C.4.3 DPX Metadata
ctest_10bpclr_CbYCrY422_lsbf_r2l_ma_norle.xml	C.4.3 DPX Metadata
ctest_10bpclr_CbYCrY422_msbf_r2l_packed_norle.xml	C.4.3 DPX Metadata
part15-j2c.xml	C.4.3 DPX Metadata
yuv422_10b_p15.xml	C.4.3 DPX Metadata
DigitalLAD.2048x1556.xml	C.4.5 ACES Metadata
SonyF35.StillLife.xml	C.4.5 ACES Metadata
syntheticChart.01.xml	C.4.5 ACES Metadata
aletest-seq1.xml	C.4.6 ALE Metadata
imf_as02_DMCVT.xml	C.4.7 Dynamic Metadata for Color Volume Transform
dark-fourcc.xml	C.5 Dark Metadata
dark-uuid.xml	C.5 Dark Metadata
multiclass-01.xml	C.6 Multiclass Metadata
multiclass-02.xml	C.6 Multiclass Metadata

D.3 Reference Bitstreams

The reference bitstreams created by the sample encoder are listed in Table D.2.

The reference bitstreams are in the test materials directory:

```
$(ROOT)/media/metadata/bitstreams
```

Each bitstream corresponds by filename to a metadata input file listed in Table D.1 and encodes the reference image `$(ROOT)/media/boxes/1280x720/rg48/boxes-1280x720-0000.rg48` listed in Annex B.2.

Table D.2 — Reference bitstreams created by the sample encoder.

Bitstream Filename	Metadata Test Case (Table D.1)
simple-c01.vc5	simple-c01.xml
simple-rdc1.vc5	simple-rdc1.xml
simple-rdc2.vc5	simple-rdc2.xml
simple-rdc4.vc5	simple-rdc4.xml
simple-rdc7.vc5	simple-rdc7.xml
simple-LOGA-stereo.vc5	simple-LOGA-stereo.xml
simple-LOGA-interlaced.vc5	simple-LOGA-interlaced.xml
simple-LOGA-hdr.vc5	simple-LOGA-hdr.xml
simple-GAMA-stereo.vc5	simple-GAMA-stereo.xml
simple-GAMA-interlaced.vc5	simple-GAMA-interlaced.xml
simple-GAMA-hdr.vc5	simple-GAMA-hdr.xml
simple-LINR-stereo.vc5	simple-LINR-stereo.xml
simple-LINR-interlaced.vc5	simple-LINR-interlaced.xml
simple-LINR-hdr.vc5	simple-LINR-hdr.xml
simple-FSLG-stereo.vc5	simple-FSLG-stereo.xml
simple-FSLG-interlaced.vc5	simple-FSLG-interlaced.xml
simple-FSLG-hdr.vc5	simple-FSLG-hdr.xml
simple-LOGC-stereo.vc5	simple-LOGC-stereo.xml
simple-LOGC-interlaced.vc5	simple-LOGC-interlaced.xml
simple-LOGC-hdr.vc5	simple-LOGC-hdr.xml
simple-PQEC-stereo.vc5	simple-PQEC-stereo.xml
simple-PQEC-interlaced.vc5	simple-PQEC-interlaced.xml
simple-PQEC-hdr.vc5	simple-PQEC-hdr.xml
simple-HLGE-stereo.vc5	simple-HLGE-stereo.xml
simple-HLGE-interlaced.vc5	simple-HLGE-interlaced.xml
simple-HLGE-hdr.vc5	simple-HLGE-hdr.xml
hero5.vc5	hero5.xml

Bitstream Filename	Metadata Test Case (Table D.1)
hero6.vc5	hero6.xml
hero7.vc5	hero7.xml
hero8.vc5	hero8.xml
karma.vc5	karma.xml
hero6+ble.vc5	hero6+ble.xml
fusion.vc5	fusion.xml
max-360mode.vc5	max-360mode.xml
max-heromode.vc5	max-heromode.xml
Image1.vc5	Image1.xml
Image2.vc5	Image2.xml
BlueSquare.vc5	BlueSquare.xml
ctest_10bpcfr_BGRA444p_lsbf_l2r_packed_rle.vc5	Corresponding XML file listed in Table D.1
ctest_10bpcfr_RGB444_lsbf_l2r_mb_norle.vc5	
ctest_10bpcfr_RGB444_msbf_l2r_ma_norle.vc5	
ctest_10bpcfr_RGBA444p_msbf_l2r_mb_rle.vc5	
ctest_10bpclr_CbYCr444_lsbf_r2l_packed_norle.vc5	
ctest_10bpclr_CbYCrA444_lsbf_r2l_packed_rle.vc5	
ctest_10bpclr_CbYCrY422_lsbf_r2l_ma_norle.vc5	
ctest_10bpclr_CbYCrY422_msbf_r2l_packed_norle.vc5	
part15-j2c.vc5	part15-j2c.xml
yuv422_10b_p15.vc5	yuv422_10b_p15.xml
DigitalLAD.2048x1556.vc5	DigitalLAD.2048x1556.xml
SonyF35.StillLife.vc5	SonyF35.StillLife.xml
syntheticChart.01.vc5	syntheticChart.01.xml
aletest-seq1.vc5	aletest-seq1.xml
imf_as02_DMCVT.vc5	imf_as02_DMCVT.xml
dark-fourcc.vc5	dark-fourcc.xml
dark-uuid.vc5	dark-uuid.xml
multiclass-01.vc5	multiclass-01.xml
multiclass-02.vc5	multiclass-02.xml

D.4 Encoded Metadata

The metadata in binary format extracted from the bitstreams created by the sample encoder from the corresponding input metadata are listed in Table D.3.

The metadata in binary format are in the test materials directory:

```
$(ROOT)/media/metadata/encoded
```

Table D.3 — Encoded metadata in binary format for conformance testing.

Encoded Metadata Filename	Bitstream File (Table D.2)
simple-c01.bin	simple-c01.vc5
simple-rdc1.bin	simple-rdc1.vc5
simple-rdc2.bin	simple-rdc2.vc5
simple-rdc4.bin	simple-rdc4.vc5
simple-rdc7.bin	simple-rdc7.vc5
simple-LOGA-stereo.bin	simple-LOGA-stereo.vc5
simple-LOGA-interlaced.bin	simple-LOGA-interlaced.vc5
simple-LOGA-hdr.bin	simple-LOGA-hdr.vc5
simple-GAMA-stereo.bin	simple-GAMA-stereo.vc5
simple-GAMA-interlaced.bin	simple-GAMA-interlaced.vc5
simple-GAMA-hdr.bin	simple-GAMA-hdr.vc5
simple-LINR-stereo.bin	simple-LINR-stereo.vc5
simple-LINR-interlaced.bin	simple-LINR-interlaced.vc5
simple-LINR-hdr.bin	simple-LINR-hdr.vc5
simple-FSLG-stereo.bin	simple-FSLG-stereo.vc5
simple-FSLG-interlaced.bin	simple-FSLG-interlaced.vc5
simple-FSLG-hdr.bin	simple-FSLG-hdr.vc5
simple-LOGC-stereo.bin	simple-LOGC-stereo.vc5
simple-LOGC-interlaced.bin	simple-LOGC-interlaced.vc5
simple-LOGC-hdr.bin	simple-LOGC-hdr.vc5
simple-PQEC-stereo.bin	simple-PQEC-stereo.vc5
simple-PQEC-interlaced.bin	simple-PQEC-interlaced.vc5
simple-PQEC-hdr.bin	simple-PQEC-hdr.vc5
simple-HLGE-stereo.bin	simple-HLGE-stereo.vc5

Encoded Metadata Filename	Bitstream File (Table D.2)
simple-HLGE-interlaced.bin	simple-HLGE-interlaced.vc5
simple-HLGE-hdr.bin	simple-HLGE-hdr.vc5
hero5.bin	hero5.vc5
hero6.bin	hero6.vc5
hero7.bin	hero7.vc5
hero8.bin	hero8.vc5
karma.bin	karma.vc5
hero6+ble.bin	hero6+ble.vc5
fusion.bin	fusion.vc5
max-360mode.bin	max-360mode.vc5
max-heromode.bin	max-heromode.vc5
Image1.bin	Image1.vc5
Image2.bin	Image2.vc5
BlueSquare.bin	BlueSquare.vc5
ctest_10bpcfr_BGRA444p_lsbf_l2r_packed_rle.bin	Corresponding bitstream file listed in Table D.2
ctest_10bpcfr_RGB444_lsbf_l2r_mb_norle.bin	
ctest_10bpcfr_RGB444_msbf_l2r_ma_norle.bin	
ctest_10bpcfr_RGBA444p_msbf_l2r_mb_rle.bin	
ctest_10bpclr_CbYCr444_lsbf_r2l_packed_norle.bin	
ctest_10bpclr_CbYCrA444_lsbf_r2l_packed_rle.bin	
ctest_10bpclr_CbYCrY422_lsbf_r2l_ma_norle.bin	
ctest_10bpclr_CbYCrY422_msbf_r2l_packed_norle.bin	
part15-j2c.bin	part15-j2c.vc5
yuv422_10b_p15.bin	yuv422_10b_p15.vc5
DigitalLAD.2048x1556.bin	DigitalLAD.2048x1556.vc5
SonyF35.StillLife.bin	SonyF35.StillLife.vc5
syntheticChart.01.bin	syntheticChart.01.vc5
aletest-seq1.bin	aletest-seq1.vc5
imf_as02_DMCVT.bin	imf_as02_DMCVT.vc5
dark-fourcc.bin	dark-fourcc.vc5
dark-uuid.bin	dark-uuid.vc5
multiclass-01.bin	multiclass-01.vc5
multiclass-02.bin	multiclass-02.vc5

D.5 Decoded Metadata

The reference metadata in XML format used for verifying conformance are listed in Table D.4.

Each file was created from the corresponding reference bitstream using the reference decoder to decode the bitstream and extract the embedded metadata to an XML file.

The files of XML metadata extracted from the reference bitstreams Annex D.3 are in the test materials directory:

```
$(ROOT)/media/metadata/decoded
```

Table D.4 — Reference metadata in XML format for verifying conformance.

Reference Metadata Filename	Bitstream File (Table D.2)
simple-c01.xml	simple-c01.vc5
simple-rdc1.xml	simple-rdc1.vc5
simple-rdc2.xml	simple-rdc2.vc5
simple-rdc4.xml	simple-rdc4.vc5
simple-rdc7.xml	simple-rdc7.vc5
simple-LOGA-stereo.xml	simple-LOGA-stereo.vc5
simple-LOGA-interlaced.xml	simple-LOGA-interlaced.vc5
simple-LOGA-hdr.xml	simple-LOGA-hdr.vc5
simple-GAMA-stereo.xml	simple-GAMA-stereo.vc5
simple-GAMA-interlaced.xml	simple-GAMA-interlaced.vc5
simple-GAMA-hdr.xml	simple-GAMA-hdr.vc5
simple-LINR-stereo.xml	simple-LINR-stereo.vc5
simple-LINR-interlaced.xml	simple-LINR-interlaced.vc5
simple-LINR-hdr.xml	simple-LINR-hdr.vc5
simple-FSLG-stereo.xml	simple-FSLG-stereo.vc5
simple-FSLG-interlaced.xml	simple-FSLG-interlaced.vc5
simple-FSLG-hdr.xml	simple-FSLG-hdr.vc5
simple-LOGC-stereo.xml	simple-LOGC-stereo.vc5
simple-LOGC-interlaced.xml	simple-LOGC-interlaced.vc5
simple-LOGC-hdr.xml	simple-LOGC-hdr.vc5
simple-PQEC-stereo.xml	simple-PQEC-stereo.vc5

Reference Metadata Filename	Bitstream File (Table D.2)
simple-PQEC-interlaced.xml	simple-PQEC-interlaced.vc5
simple-PQEC-hdr.xml	simple-PQEC-hdr.vc5
simple-HLGE-stereo.xml	simple-HLGE-stereo.vc5
simple-HLGE-interlaced.xml	simple-HLGE-interlaced.vc5
simple-HLGE-hdr.xml	simple-HLGE-hdr.vc5
hero5.xml	hero5.vc5
hero6.xml	hero6.vc5
hero7.xml	hero7.vc5
hero8.xml	hero8.vc5
karma.xml	karma.vc5
hero6+ble.xml	hero6+ble.vc5
fusion.xml	fusion.vc5
max-360mode.xml	max-360mode.vc5
max-heromode.xml	max-heromode.vc5
Image1.xml	Image1.vc5
Image2.xml	Image2.vc5
BlueSquare.xml	BlueSquare.vc5
ctest_10bpcfr_BGRA444p_lsbf_l2r_packed_rle.xml	Corresponding bitstream file listed in Table D.2
ctest_10bpcfr_RGB444_lsbf_l2r_mb_norle.xml	
ctest_10bpcfr_RGB444_msbf_l2r_ma_norle.xml	
ctest_10bpcfr_RGBA444p_msbf_l2r_mb_rle.xml	
ctest_10bpclr_CbYCr444_lsbf_r2l_packed_norle.xml	
ctest_10bpclr_CbYCrA444_lsbf_r2l_packed_rle.xml	
ctest_10bpclr_CbYCrY422_lsbf_r2l_ma_norle.xml	
ctest_10bpclr_CbYCrY422_msbf_r2l_packed_norle.xml	
part15-j2c.xml	part15-j2c.vc5
yuv422_10b_p15.xml	yuv422_10b_p15.vc5
DigitalLAD.2048x1556.xml	DigitalLAD.2048x1556.vc5
SonyF35.StillLife.xml	SonyF35.StillLife.vc5
syntheticChart.01.xml	syntheticChart.01.vc5
aletest-seq1.xml	aletest-seq1.vc5
imf_as02_DMCVT.xml	imf_as02_DMCVT.vc5

Reference Metadata Filename	Bitstream File (Table D.2)
dark-fourcc.xml	dark-fourcc.vc5
dark-uuid.xml	dark-uuid.vc5
multiclass-01.xml	multiclass-01.vc5
multiclass-02.xml	multiclass-02.vc5

Annex E Additional Elements (Informative)

E.1 Repository

A reference implementation that is believed to be compliant with the VC-5 essence standard is provided as a companion software element to this document. The software and tools, test images, and sample bitstreams referred to in this document are in the public repository at the following URL:

https://github.com/SMPTE/rp2073-2-a.git

The software version corresponding to this document revision is tagged as SMPTE-RP-2073-2-2022.

E.2 Root Directory

Throughout this document, the value of the symbol `$(ROOT)` is the directory into which the test materials are cloned from the repository.

Bibliography

SMPTE ST 268-1:2014 File Format for Digital Moving-Picture Exchange (DPX)

SMPTE ST 268-2:2018 Digital Moving-Picture Exchange (DPX) – Format Extensions for High Dynamic Range and Wide Color Gamut

SMPTE ST 377-1:2019 Material Exchange Format (MXF) – File Format Specification

SMPTE ST 2065-4:2013 ACES Image Container File Layout

SMPTE ST 2073-1:2017 VC-5 Video Essence. Part 1: Elementary Bitstream

SMPTE ST 2073-3:2015 VC-5 Video Essence. Part 3: Image Formats

SMPTE ST 2073-4:2015 VC-5 Video Essence. Part 4: Subsampled Color Difference Components

SMPTE ST 2073-5:2015 VC 5 Video Essence. Part 5: Layers

SMPTE ST 2073-6:2015 VC-5 Video Essence. Part 6: Sections

SMPTE ST 2073-7:2022 VC-5 Video Essence. Part 7: Metadata

SMPTE ST 2094-2:2017 Dynamic Metadata for Color Volume Transform – KLV Encoding and MXF Mapping

ISO/IEC 9899:2011 Programming Language C (C11)

ISO/IEC 13818-2:2013 Information technology – Generic coding of moving pictures and associated audio information -- Part 2: Video

IETF RFC 4180 Common Format and MIME Type for Comma-Separated Values (CSV) Files
https://tools.ietf.org/html/rfc4180

IETF RFC 4648 The Base16, Base32, and Base64 Data Encodings
https://tools.ietf.org/html/rfc4648#section-5

IETF RFC 8259 The JavaScript Object Notation (JSON) Data Interchange Format
https://tools.ietf.org/html/rfc8259

W3C Recommendation Extensible Markup Language (XML) 1.0 (Fifth Edition), 26 November 2008
Available from: https://www.w3.org/TR/xml

John K. Ousterhout and Ken Jones, Tcl and the Tk Toolkit (2nd Edition), Addison-Wesley, 2009

Python: https://www.python.org/

GPMF Parser: https://github.com/gopro/gpmf-parser.git

XMP Toolkit SDK: https://github.com/adobe/XMP-Toolkit-SDK.git

ExifTool: https://exiftool.org/

Informative reference implementation for ST 268-2 (HDR extensions to DPX):
https://github.com/SMPTE/smpte-31fs-hdrdp

OpenEXR: https://www.openexr.com/

AMPAS Academy Color Encoding System Developer Resources: https://github.com/ampas/aces-dev