International Standard

ISO/IEC 19566-10

First edition
2024-12

Information technology — JPEG Systems —

Part 10: Reference software

Reference number
ISO/IEC 19566-10:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO 19566 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The JPEG Universal Metadata Box Format (JUMBF) provides a mechanism to embed and refer generic metadata in JPEG files. Specific content types can be assigned to identify the specific type of the embedded metadata.

This document describes a reference software implementation that handles JUMBF data according to the Content Types specified in ISO/IEC 19566-5. Initially, the JUMBF reference dataset is presented, consisting of standalone JUMBF files as well as JPEG-1 encoded images. In addition, the respective validation procedure is specified which enables the validation of candidate implementations of ISO/IEC 19566-5. Finally, a detailed analysis of the Java reference implementation of ISO/IEC 19566-5 is presented, demonstrating the software design which is followed to support the JUMBF data model.

Information technology — JPEG Systems —

Part 10:
Reference software

1 Scope

This document specifies a reference software implementation of ISO/IEC 19566-5. The reference software is accompanied with a reference dataset which provides an extensive list of the various JUMBF data structures specified in ISO/IEC 19566-5.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 19566-5, *Information technologies — JPEG systems — Part 5: JPEG universal metadata box format (JUMBF)*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org

3.1.1
codestream
compressed image data representation that includes all necessary data to allow a (full or approximate) reconstruction of the sample values of a digital image

3.2 Abbreviated terms

APP11	application marker 11: JPEG XT extension marker
CBOR	concise binary object representation
CLI	command line interface
CSV	comma separated values
GUI	graphical user interface
IoC	inversion of controls
ISOBMFF	ISO base media file format

IV	initial value
JPEG	joint photographic experts group
JPEG-1	image complying to ISO/IEC 10918-1
JP2C	JPEG contiguous codestream
JSON	JavaScript[1)] object notation
JUMBF	jpeg universal metadata box format
POJO	plain old java object
UUID	universally unique identifier
XML	extensible markup language

4 Reference software

4.1 Purpose

The use of the reference software is not required for making an implementation of parser or generator in conformance to any of the parts of the ISO/IEC 19566 series. Requirements established in all parts of the ISO/IEC 19566 series take precedence over the behaviour of the reference software.

4.2 Examples of use

This subclause enumerates possible uses of the reference implementations presented in the annexes:

a) Sample parser. Users can use the reference implementations to inspect, even through visualization, the structure and contents of the JUMBF data model as specified in ISO/IEC 19566.

b) Sample generator. Users could use the reference implementations to create JUMBF files that could be used to facilitate the development of applications that take advantage of the benefits of JUMBF specification.

c) Provide anchor implementations for development purposes. JUMBF data model developers could study the reference implementations presented in the annexes in order to gain better insight on the algorithms as well as on how JUMBF Boxes are interconnected.

d) Provide anchor implementations to test possible conformant software. This facilitates developers to have an existing implementation act as a ground truth which will assist them assess the validity of their own implementations.

The lack of detection of any conformance violation by any reference software implementation should not be considered as a definite proof that the codestream under testing conforms to all constraints required for it to be conforming to one of the parts of the ISO/IEC 19566 series. Similarly, the computation resource characteristics in terms of program or data memory usage, execution speed, etc. of sample software encoder or decoder implementations shall not be construed as a representative of the typical, minimal or maximal computational resource characteristics to be exhibited by implementations of some parts of the ISO/IEC 19566 series.

1) JavaScript™ is the trademark of a product supplied by Oracle® Corporation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

4.3 Warranty disclaimer

Regardless of any and all statements made herein or elsewhere regarding the possible uses of the reference software, the following disclaimers of warranty apply to the provided reference software implementations:

— ITU, ISO and IEC disclaim any and all warranties, whether express, implied, or statutory, including any implied warranties of merchantability or of fitness for a particular purpose.

— In no event shall the contributor(s) or ITU, ISO or IEC be liable for any incidental, punitive or consequential damages of any kind whatsoever arising from the use of these programs.

— This disclaimer of warranty extends to the user of these programs and the user's customers, employees, agents, transferees, successors, and assignees.

— ITU, ISO and IEC do not represent or warrant that the software is free of infringements of any patents.

— Commercial applications of ITU-T Recommendations and ISO/IEC International Standards, including shareware, may be subject to royalty fees to patent holders.

4.4 General

The rest of the document describes several reference software implementations. The reference software for this document is available at https://standards.iso.org/iso-iec/19566-10/ed-1/en/. Reference software implementations do not intend to be unique. Therefore, some parts of JPEG Systems can have more than one implementation. On the other hand, reference software implementations need to be validated, from functionality and interoperability points of view. Hence, Annex A describes the mechanism followed for validation of the JUMBF reference software. Annex B describes an implementation of reference software for JUMBF (as specified in ISO/IEC 19566-5). The implementations presented in the subsequent annexes are summarized in Table 1, including information related to the parts of the ISO/IEC 19566 series they cover, the provided functionalities as well as the technology used.

Table 1 — Reference software implementations for the ISO/IEC 19566 series

Annex	Software	ISO/IEC 19566 parts	Decoder	Encoder	Technology
Annex B	jumbf-2.0 library	ISO/IEC 19566-5	Yes	Yes	Java[a]

[a] Java™ is the trademark of a product supplied by Oracle® Corporation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Annex A
(informative)

Validation of ISO/IEC 19566-5 (JUMBF) reference software

A.1 General

This subclause describes the validation process that is followed in order to verify the correctness of ISO/IEC 19566-5 reference software implementations. The validation procedure deals with parser and generator implementations separately. Along with the validation procedure, the JPEG Systems reference dataset is included. In principle, any reference implementation presented in this document can parse/generate the reference files of the related part of the ISO/IEC 19566 series.

In general, the aim of the JPEG Systems reference dataset is to address all the available JUMBF Content Types specified in the scope of the ISO/IEC 19566 series and covering cases of standalone JUMBF files, but also embedding JUMBF Boxes in supporting image file formats. The JPEG Systems reference dataset is split into multiple subdirectories, each of which corresponds to a specific part of the ISO/IEC 19566 series that specifies JUMBF data structures. All the available parts of the ISO/IEC 19566 series and the respective JUMBF Content Types that are specified in each one of them is listed in Table A.1.

Table A.1 — JUMBF Content Types as defined in the ISO/IEC 19566 series

Directory name	JUMBF Content Types
Part 5	XML, JSON, JP2C, CBOR, UUID, Embedded File
Part 4	Protection, Replacement
Part 6	JPEG 360
Part 7	JLINK
Part 8	JPEG Snack

A.2 JUMBF reference dataset

The first version of the JPEG Systems reference dataset — which is attached along with this document — consists of exactly one directory, namely the JUMBF reference dataset. This directory consists of an exhaustive list of standalone JUMBF files covering ISO/IEC 19566-5. All these files cover plenty of combinations for the available JUMBF Description box attributes and Content Types in ISO/IEC 19566-5. A detailed description about the information of each of the available files is given in the reference dataset description, a csv file which accompanies the dataset and signals the contents of the Description box and the respective Content Boxes. The columns of the reference dataset description file for JUMBF reference dataset are presented in Table A.2. To specify the content of a JUMBF Box (e.g., the XML payload of an XML Content type JUMBF box), a set of input files is included in the dataset. The file name of each of the input documents is referenced through the csv file in the respective columns. In scope of ISO/IEC 19566-5 the contents of a JUMBF Box can be an XML serialized content, a JSON serialized content, a CBOR serialized content, a JPEG codestream or a generic bytestream. In principle, with the reference dataset description file it is possible to define any combination of JUMBF data. Specifically, columns A-K refer to all the various combinations related to the Description box data model. Columns L and M specify the expected number and content (i.e., input file) of the Content Box for each JUMBF Box defined in ISO/IEC 19566-5. Next, columns N-O correspond to the specific fields defined in UUID Content type JUMBF box, while columns P-R correspond to those of Embedded File Content type JUMBF box. Finally, column S points to the file name of the generated file where the JUMBF data is going to be stored. If a column is not applicable for a specific JUMBF structure, “NULL” value is used. Each line of the csv defines a JUMBF Box that is stored either as a standalone file or embedded in a host image which is encoded using one of the available JPEG encoding formats. Normally, each line

corresponds to a unique file. However, it is also possible for multiple lines to specify a common value in column S. This means that multiple JUMBF Boxes are concatenated to a single file.

Table A.2 — Definition of the JUMBF reference dataset description csv file.

Csv column number	Csv column name	Csv column description	NULL value allowed?
A	Test Id	Name of the specific scenario described in the row	No
B	Content Type (UUID)	The JUMBF Content Type of the JUMBF Box to be generated.	No
C	Host JPEG Image	The name of the JPEG Encoded image that will host the generated JUMBF Box. If NULL, the resulted JUMBF Box will be stored in a separate file (i.e., JUMBF standalone file) with the .jumbf extension.	Yes
D	LBox	Value of the LBox attribute of the JUMBF Box header. Available values: 0 (i.e., Read until the end of file), 1 (i.e., the size in bytes of the box is expressed in the XLBox attribute of the JUMBF Box header), NULL (The actual size in bytes of the JUMBF Box is stored in LBox JUMBF Box header).	Yes
E	Is Requestable?	Boolean value included in the Description box. It corresponds to the "Requestable" field as specified in ISO/IEC 19566-5.	No
F	Label	String value included in the Description box. It corresponds to the "Label" field as specified in ISO/IEC 19566-5.	Yes
G	ID	Numerical value included in the Description box. It corresponds to the "Id" field as specified in ISO/IEC 19566-5.	Yes
H	SHA256HASH (file name)	The name of the file pointing to the bytestream value included in the Description box. It corresponds to the "SHA256Hash" field as specified in ISO/IEC 19566-5.	Yes
I	Contains multiple private fields?	Boolean value that specifies whether there is a private field included in the requested Description box. If set to FALSE, it means that either there is no Private field or there is a single ISOBMFF Box. If set to TRUE, it means that the Private field consists of a 'priv' box.	No
J	Private Field (JUMBF Box file name)	The name of the JUMBF standalone file that contains the bytestream that corresponds to the contents of the private field of the requested Description box.	Yes
K	Padding (Number of bytes)	The number of bytes allocated for the padding box content. If set to 0 then no Padding box is added.	No
L	Number of Content Boxes	The number of Content Boxes included in the requested JUMBF Box.	No
M	Content Box (file name)	The name of the file that corresponds to a Content Box of the requested JUMBF Box. For instance, if the requested JUMBF Box is of XML Content type, then this column could specify a "example.xml" file, pointing to the contents of such JUMBF Box.	No

Table A.2 *(continued)*

Csv column number	Csv column name	Csv column description	NULL value allowed?
N	UUID	The 16-byte UUID field as specified in the UUID Content type in ISO/IEC 19566-5.	Yes
O	Data	The name of the file pointing to the bytestream included in a UUID Content type JUMBF box.	Yes
P	Media Type	String value included in the Embedded File Description box. It corresponds to the “Media Type” field as specified in ISO/IEC 19566-5.	Yes
Q	Filename	String value included in the Embedded File Description box. It corresponds to the “FILE NAME” field as specified in ISO/IEC 19566-5.	Yes
R	Is Externally Referenced?	Boolean value included in the Embedded File Description box. It corresponds to the “External Reference” field as specified in ISO/IEC 19566-5.	Yes
S	Expected file (file name)	The name of the generated file that contains the JUMBF Box described in the Test ID of the respective entry.	No

Finally, apart from the reference dataset and its description file, the JUMBF reference dataset contains another csv file which corresponds to a reference report. The reference report contains information that a parser information can extract from a JUMBF file in the context of ISO/IEC 19566-5. The columns of this csv file are different from the reference dataset description file and their definition is listed in Table A.3.

Table A.3 — Definition of the JUMBF reference dataset report csv file.

Csv column number	Csv column name	Csv column description	NULL value allowed?
A	File name	Name of the parsed file which contains JUMBF-related information	No
B	Standalone file?	Boolean value that specifies whether the parsed file contains JUMBF information embedded in a (host) JPEG image or contains JUMBF data only.	No
C	LBox	Numerical value of LBox attribute in JUMBF header.	No
D	XLBox	Numerical value of XLBox attribute in JUMBF header.	Yes
E	Content Type UUID	String value specifying the UUID field included in the parsed Description box.	No
F	Description box Toggle	Numerical value included in the parsed Description box. It corresponds to the “Toggle” field as specified in ISO/IEC 19566-5.	No
G	Is requestable	Boolean value that specifies whether the parsed JUMBF Box is requestable. It corresponds to the “Requestable” field as specified in ISO/IEC 19566-5.	No
H	Label	String value specifying the Label field included in the parsed Description box. It corresponds to the “Label” field as specified in ISO/IEC 19566-5.	Yes
I	ID	Numerical value specifying the ID field included in the parsed Description box. It corresponds to the “ID” field as specified in ISO/IEC 19566-5.	Yes

Table A.3 *(continued)*

Csv column number	Csv column name	Csv column description	NULL value allowed?
J	SHA256 Hash	String value specifying the HEX encoded SHA-256Has value of the parsed Description box. It corresponds to the "SHA256Hash" field as specified in ISO/IEC 19566-5.	Yes
K	Private field Exists?	Boolean value specifying whether a private field is included in the parsed Description box.	No
L	UUID box UUID field	String value specifying the UUID field included in the parsed UUID box. The 16-byte UUID field is specified in the "UUID Content type" subclause of ISO/IEC 19566-5.	Yes
M	Embedded File Description box Toggle	Numerical value specifying the toggle of the parsed Embedded File Description box. It corresponds to the "Toggle" field as specified in the "Embedded File Content type" subclause of ISO/IEC 19566-5.	Yes
N	Embedded File Description box Media Type	String value specifying the media type of the parsed Embedded File Description box. It corresponds to the "Media Type" field as specified in the "Embedded File Content type" subclause of ISO/IEC 19566-5.	Yes
O	Embedded File Description box File Name	String value specifying the file name of the parsed Embedded File Description box. It corresponds to the "File name" field as specified in the "Embedded File Content type" subclause of ISO/IEC 19566-5.	Yes
P	Embedded File Description box Content Referenced Externally	Boolean value specifying whether the parsed Embedded File Content type JUMBF box is referenced externally or it is embedded in the current box. It corresponds to the "External file" field as specified in the "Embedded File Content type" subclause of ISO/IEC 19566-5.	Yes
Q	Content Box TBox field	String value of four characters specifying the 4-byte-type value of the Content Box of the parsed JUMBF Box (e.g. 'json', 'bidb').	Yes
R	Content Box Size	Numerical value specifying the number of bytes of the Content Box of the parsed JUMBF Box.	Yes
S	JUMBF Box Padding box Size	Numerical value specifying the number of bytes included in the Padding box of the parsed JUMBF Box.	No

A.3 Validation procedure

A.3.1 General

Given the reference dataset along with its description and report files it is possible to define a validation procedure that parser and generator implementations could use in order to verify their conformance with ISO/IEC 19566-5. Specifically, this section uses the dataset introduced in A.2 and defines the validation procedures for reference implementations in scope of ISO/IEC 19566-5. The validation procedure could be easily generalized for all parts of the ISO/IEC 19566 series mentioned in the annexes.

A.3.2 Parser validation

Figure A.1 shows the procedure of validating a candidate parser implementation. For a parser implementation to check its conformance with ISO/IEC 19566-5, it shall be able to provide a report which describes the contents of a JUMBF Box following the structure presented in the reference report.

Initially, the JUMBF reference dataset is provided to the candidate implementation which parses all the files and generates a report. The generated report is first sorted by the first column (i.e., the name of the JUMBF file) and it is compared with the JUMBF reference dataset report. Provided that the two reports are identical, the candidate parser implementation is considered conformant to ISO/IEC 19566-5.

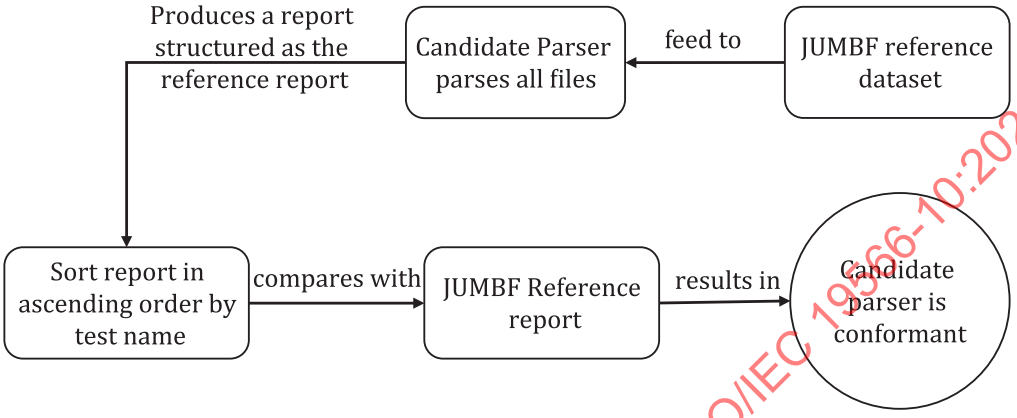

Figure A.1 — Steps to show that a candidate parser implementation conforms to ISO/IEC 19566-5

A.3.3 Generator validation

Figure A.2 shows the procedure of validating a candidate generator implementation. For a generator implementation to check its conformance with ISO/IEC 19566-5 it shall be able to reconstruct the available JUMBF reference dataset, given the respective reference dataset description file.

Initially, the JUMBF reference dataset description is provided to the generator implementation along with all the input files that are referenced in it. The candidate implementation should parse the csv file and generate the corresponding JUMBF data. Finally, the generated dataset is submitted for cross-checking with the reference dataset. Assuming that all files are identical, the candidate generator implementation is considered conformant to ISO/IEC 19566-5.

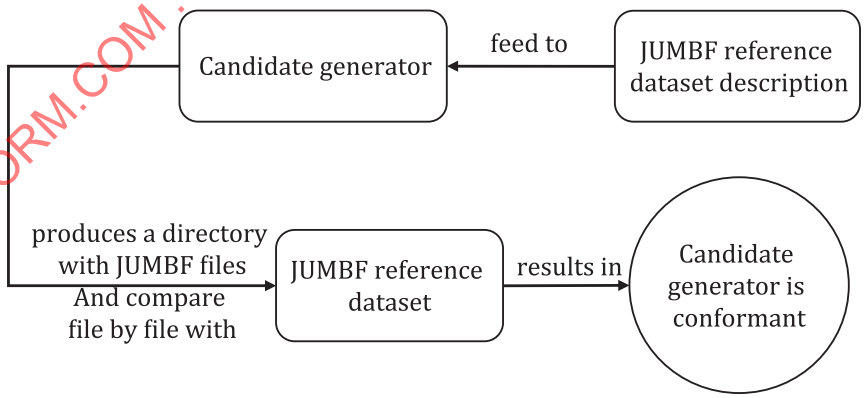

Figure A.2 — Steps to show that a candidate generator implementation conforms to ISO/IEC 19566-5

Annex B
(informative)

JUMBF reference software: Java implementation

B.1 General

This annex describes a Reference software implementation (identified as Implementation 1) for ISO/IEC 19566-5 JPEG Systems — Part 5: JPEG universal metadata box format (JUMBF) ISO/IEC 19566-5. The software library is called jumbf-2.0 and it is the main part of a multi-module project, namely jpeg-systems-1.0, whose goal is to support all the JUMBF Boxes presented in the JPEG Systems specifications. It is written in Java[3] and it allows parsing and generating JUMBF data according to ISO/IEC 19566-5. The design of this software aims to be extended in order to support JUMBF structures from other ISO/IEC 19566 parts. The source code is included under the directory "Reference Implementation/annex-b/java-reference-implementation".

This annex provides background information on the software design approach followed for the development of this reference software for JPEG Universal Metadata Box Format (JUMBF). In addition, it provides details on how to compile the jumbf-2.0 library and how it could be used by other Java applications in order to successfully handle JUMBF data. Subclause B.2 defines the hierarchical software design which translates the JUMBF model presented in ISO/IEC 19566-5, into a set of Java classes in a structured and future-proof manner. Next, subclause B.2.4 presents the requirements and the third-party dependencies required in order to compile and use the jumbf-2.0 library. Finally, subclause B.3 demonstrates how two different applications use the library in order to provide an interface which allows the users to interact with JUMBF data.

B.2 Software design

B.2.1 General

The aim of the jumbf-2.0 library is to provide the means to support the manipulation of JUMBF data as specified in ISO/IEC 19566-5. Since there are additional JUMBF Boxes defined in other ISO/IEC 19566 parts, the jumbf-2.0 library is a module of a broader project where additional libraries extend its functionalities to define other JUMBF structures. The jumbf-2.0 library, as all the other modules defined in this project, cover a specific ISO/IEC 19566 part including the amendments and revised editions that have been issued. In order for the libraries to be in line with the JPEG activities, the versioning system of the project follows that of the ISO/IEC 19566 parts. Thus, for the JUMBF reference software, the jumbf-2.0 library supports the model and functionalities specified in scope of ISO/IEC 19566-5 that corresponds to ISO/IEC 19566-5:2023.

Following the logical structure between the various ISO/IEC 19566 parts, jumbf-2.0 is the main library that all other libraries of the project should rely on. In fact, the jumbf-2.0 library provides the means to generate and parse information that is stored in JUMBF format. The library is written in Java and it uses Spring[2] Framework[5]. It provides an architecture where the JUMBF model is implemented in a way that other libraries could inherit the JUMBF structure characteristics by simply adding jumbf-2.0 library as a dependency. This allows the architecture to be minimal and extendable.

As specified in ISO/IEC 19566-5, it is possible to store JUMBF data as standalone files or inside a host image by embedding the boxes in APP11 markers. Regarding the first case, the jumbf-2.0 library provides the classes to parse and generate JUMBF data directly from standalone files using the CoreParserService and CoreParserGenerator classes. Regarding the generation of standalone JUMBF data, the file extension ".jumbf" is used, corresponding to the concatenation of ISO Base Media File Format boxes. To address the second case,

2) Spring® is the trademark of a product supplied by Broadcom Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

the library implements the JpegCodestreamParserService and JpegCodestreamGeneratorService classes. With these two classes, the library is able to handle JUMBF data embedded in JPEG-1 encoding images. The logic implemented in these classes provides the means to support the ISO Base Media File Format specified in ISO/IEC 18477-3. In essence, these classes traverse through the entire JPEG codestream and focus only on the APP11 markers that are related to JUMBF information.

The rest of the section describes the implementation of the classes that are defined to support specifically the ISO/IEC 19566-5. The core concept in this data model is a Box. To describe a Box structure in the jumbf-2.0 library two classes need to be specified: an Entity class and a Service class. An Entity class contains the information concerning the fields that are defined in a particular JUMBF Box. Next, the functionalities to parse and generate a particular box, are included in the respective Service class. This allows for a better separation of concerns in our software. The class hierarchies for Entity and Services classes are defined in subclause B.2.2. Finally, once all JUMBF Boxes are defined, it is possible to define the JUMBF Content Types which are supported in ISO/IEC 19566-5. The definition of the hierarchy of Content Type classes is presented in subclause B.2.3.

B.2.2 JUMBF Box definition: Entity and Service classes

This section presents the two class hierarchies required for the implementation of any JUMBF Box, namely the Entity and the Service class hierarchies. Figure B.1 presents the class hierarchy for all the Entity classes. At the top of the figure, the BoxInterface interface defines two methods that each Box structure should support: “Get the Box Type Id” and “Get the size of the Box”. In other words, each Entity class should be able to specify its ISOBMFF ISO/IEC 18477-3 TBox and its size including the length of its headers. Next, a BmffBox abstract class is defined, implementing the BoxInterface. By definition, the first bytes of any Box that is defined in ISO/IEC 19566-5, correspond to the ISOBMFF header. Specifically, these bytes signal:

— The Length of the box (LBox): 4 Bytes,

— The Type of the box (TBox): 4 Bytes,

— (Optional) The box length extension (XLBox): 8 Bytes.

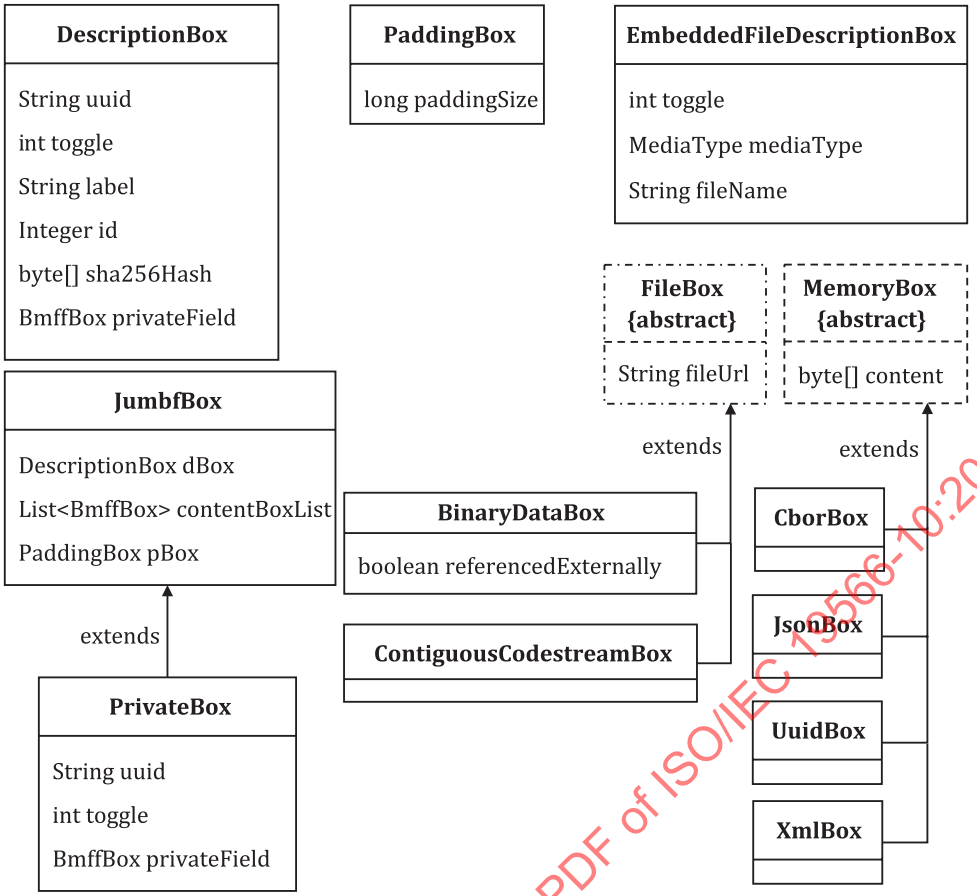

Figure B.1 — Entity class hierarchy in jumbf-core-2.0

Subsequently, a DescriptionBox class is implemented in order to cover the Description box definition. This class is a Plain Old Java Object (POJO) that specifies fields as defined in the second edition of ISO/IEC 19566-5. The final field of a DescriptionBox class is of type BmffBox and it is intended to describe a private field as defined in ISO/IEC 19566-5:2023. A private field can be any ISOBMFF box, or a Private Box the definition of which is covered in the PrivateBox class. Furthermore, the second edition of ISO/IEC 19566-5 defines a Padding Box structure (implemented in the PaddingBox Entity class) which allows for size manipulation of a JUMBF box. With these Box definitions, it is possible to implement the Entity class of a generic JUMBF Box. In the jumbf-2.0 library, it is defined as a set of fields consisting of i. one DescriptionBox field, ii. a list of BmffBox elements corresponding to the Content Boxes of a JUMBF Box and iii. a field of type PaddingBox. The remaining box structures of the 2nd edition of ISO/IEC 19566-5 are defined. These boxes are used for storing textual and binary data according to the Content Type of each JUMBF Box.

NOTE The validity of the content of each content box is out of scope of the functionalities of the jumbf-2.0 library. For example, the library does not verify that the content of a JsonBox class is a valid JSON representation. It is up to the application that uses this library to evaluate the content of a Content Box.

To benefit from code-reusability, jumbf-2.0 library defines two abstract classes (depicted as dashed rectangles) that distinguish how the data of a content box is handled: a FileBox and a MemoryBox. The former one defines a String field “fileUrl” which contains the absolute path to the file containing the contents that should be included in that particular Content Box. The latter abstract box defines a byte array that allows the storage of the entire information of a Content Box in the buffer.

So far, with the class hierarchy of the Entity classes it is possible to define the structure of any Box that is specified. However, the jumbf-2.0 library provides the means to parse and generate each Box. This is achieved through the Service class hierarchy, which provides the means to parse/generate each structure that has been defined in the Entity classes. Figure B.2 presents the Service class hierarchy at the top of which the BoxService interface is defined. This interface is implemented by every Service class and dictates every class to provide a method for writing and for parsing the bytes related to that Box. The first class is

the BmffBoxService abstract class. This class implements the two aforementioned methods (i.e., parse and generate) for the bytestreams that correspond to the ISOBMFF fields, as defined in the respective BmffBox class. Each of the Service classes that is specified in Figure B.2 inherits from the BmffBoxService as before handling the internals of each structure it is essential that the ISOBMFF fields are processed. Each Service class is implemented as Spring Bean[6], a core object of the Spring Framework. It is managed (instantiated, assembled) by the Spring Inversion of Controls (IoC) container. With Spring Beans, it is possible to decouple the dependencies between various classes but also to provide algorithms which are valid for Beans that haven't been defined in the jumbf library; instead, Spring Beans could be defined and loaded by a child library that is dependent on the jumbf library.

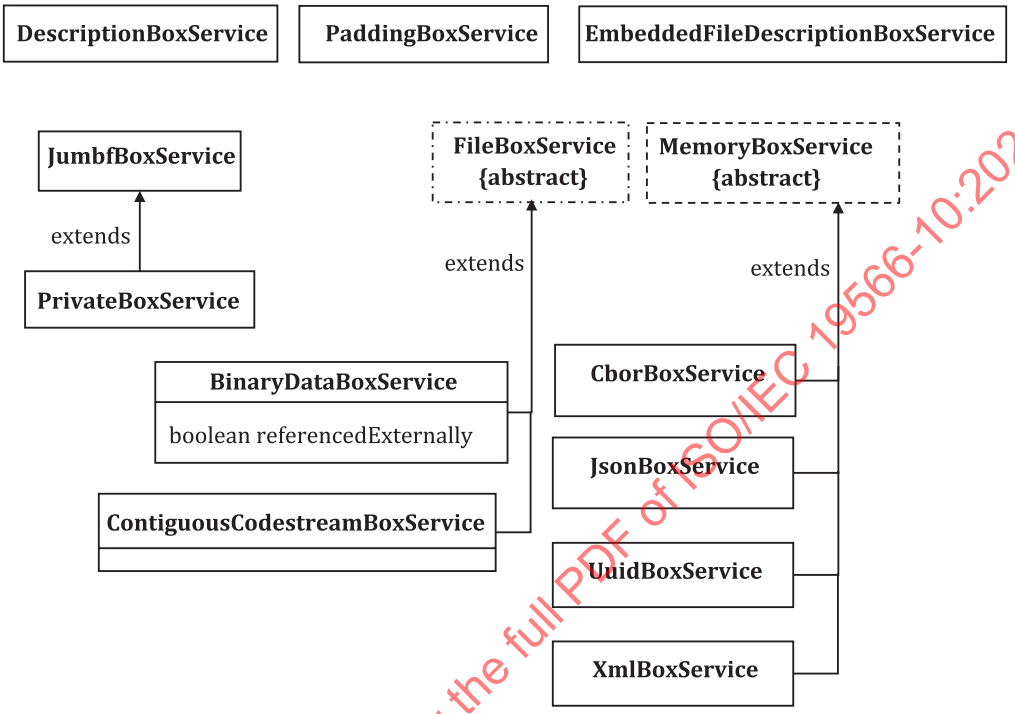

Figure B.2 — Service class hierarchy in jumbf-core-2.0

Having defined all the dependent services, it is worth showcasing the algorithm that handles any JUMBF Box definition. The implemented algorithm for parsing and generating JUMBF data is future-proof in the sense that it defines generic steps which allow external libraries to define their own JUMBF structures being required to handle the JUMBF data model from scratch. From now on, the term "handle" will be used in order to cover both parsing and generating activities as the algorithm is similar in both cases. First of all, since JumbfBoxService class is inheriting from the BmffService class, the first bytes that are going to be handled are the ones related to ISOBMFF headers. Next, the algorithm calls the DescriptionBoxService in order to handle the DescriptionBox. Based on the UUID field of the Description box the algorithm calls a discovery process, namely ContentTypeDiscoveryManager. This class is responsible for assigning a new object which belongs to a new family of classes that implement the interface ContentType. The details of this class hierarchy are presented in section B.2.3. In brief the ContentType classes provide the means to handle a set of Boxes based on the definition of a Content Type JUMBF Box. The ContentType classes are implemented as Spring Beans. Upon initialization of the context of the application that uses jumbf-2.0 library, the ContentTypeDiscoveryManager collects all the Spring Beans that extend the ContentType interface. This provides the capability for additional libraries who depend on jumbf-2.0 library to define their own ContentType classes and use their own Entity and Service classes without getting involved into the internals of how a JUMBF Box is structured. Once the ContentType class has handled the byte stream corresponding to the Content Box(es) of a JUMBF Box, the JumbfBoxService class covers the cases where there is a Padding box to be handled. Provided that there are sufficient bytestreams allocated for the particular JUMBF Box, the PaddingBoxService is invoked.

B.2.3 Content Type classes

As mentioned above, apart from the two class hierarchies presented to define all the box structures defined in ISO/IEC 19566-5, the ContentType class hierarchy is implemented in order to combine the correct box classes and shape a supported JUMBF Box. In ISO/IEC 19566-5, there are two categories of ContentType classes, similar to the two different structures of Content Type JUMBF Boxes: JUMBF Boxes with one or two Content Boxes. JSON, XML, CBOR, UUID and JP2C Content type JUMBF boxes fall in the former category while Embedded File Content type JUMBF box falls into the latter one as it consists of two Content Boxes: an Embedded File Description box and a Binary Data box.

Figure B.3 presents the entire class hierarchy that composes the Content Boxes as defined in ISO/IEC 19566-5. Each of the classes implemented inherit from the ContentType interface which obliges the classes to provide a UUID corresponding to the Content Type UUID that the class addresses. In addition, it provides the algorithms to use the required Service classes which provide the required Boxes. For instance, in the case of a JSON Content type JUMBF box, the JsonContentType class will only use the JsonBoxService Bean to handle the JSON Box. On the contrary, for the Embedded File Content type JUMBF box, first the EmbeddedFileDescriptionService is invoked, followed by the BinaryDataBoxService.

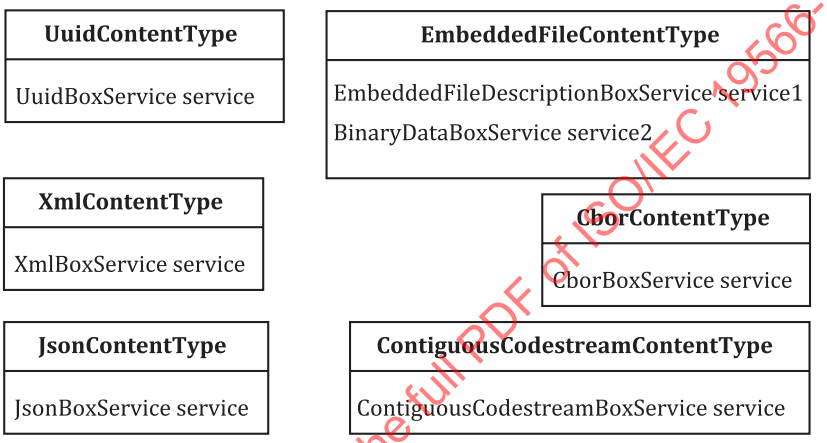

Figure B.3 — Content Type class hierarchy in jumbf-core-2.0

B.2.4 Software requirements

The entire multi-module project is built using Java 11[2], an object-oriented programming language. Secondly, maven 3.9.0 [3] is used to manage the third party library dependencies of the project as well as the interdependencies between all the libraries inside the project. All the dependencies are specified in pom. xml files located in the project's home directory and in the respective submodules' directories. Maven also facilitates the entire lifecycle of the project starting from the development and the testing up to packaging.

With the aforementioned two tools, any interested developer has the ability to compile and use the jumbf library. Specifically, once in the home directory of the project, the command "mvn clean install" will compile the source code and install it in the local repository as a library that can be used by other applications. During the first execution of this maven command, all the necessary dependencies of the project should be downloaded. Some of the dependencies are considered as global, in the sense that all submodules of the project will inherit them. In addition to these dependencies, there might be the case where a set of dependencies is required for a specific submodule. The entire list of the third-party libraries — including their transient dependencies — is listed in Table B.1.

Table B.1 — Library dependencies for jumbf-2.0 library.

No	Group Id	Artifact Id	Version	Comment	Licence
1	SpringFramework	Spring Core	5.3.25	Global dependency	Apache License, Version 2.0
2	SpringFramework	Spring Context	5.3.25	Global dependency	Apache License, Version 2.0
3	SpringFramework	Spring Beans	5.3.25	Global dependency	Apache License, Version 2.0
4	SpringFramework	Spring Web	5.3.25	Global dependency	Apache License, Version 2.0
6	SpringFramework	Spring JCL	5.3.25	Global, transient dependency	Apache License, Version 2.0
7	SpringFramework	Spring AOP	5.3.25	Global, transient dependency	Apache License, Version 2.0
8	SpringFramework	Spring Expression	5.3.25	Global, transient dependency	Apache License, Version 2.0
9	FasterXML Jackson Dataformat	Jackson Dataformat: CBOR	2.13.2	jumbf-2.0 library dependency	Apache License, Version 2.0
10	FasterXML Jackson Core	Jackson Databind	2.13.2	jumbf-2.0 library, transient dependency	Apache License, Version 2.0
11	FasterXML Jackson Core	Jackson Annotations	2.13.2	jumbf-2.0 library, transient dependency	Apache License, Version 2.0
12	FasterXML Jackson Core	Jackson Core	2.13.2	jumbf-2.0 library, transient dependency	Apache License, Version 2.0

Based on the summarized table presented above, dependencies 1-8 are used in order to provide the Spring Beans framework that allows the decoupling of dependencies between the various Service and ContentType classes that are defined in subclause B.2. In addition, with the Spring Beans, it is possible to implement the discovery algorithm which is responsible for assigning the suitable ContentType class in order to handle (i.e., parse or generate) the Content Boxes of a JUMBF Box according to its Description Box UUID field. Next, dependencies 9-14 are necessary in order to define a testing environment which facilitates and speeds up the development process, ensuring that all the functionalities are working as expected. Finally, jumbf-2.0 requires dependencies 15-18 in order to support the serialization/deserialization of CBOR information.

The licences of all the aforementioned dependencies are listed in the ThirdPartyNotices.txt files, located in the home directory of the entire project or the respective submodule(s) depending on whether the dependency is global or not.

B.3 Applications

B.3.1 General

This section showcases how two different Java applications use the Java multi-module project, and specifically the jumpf-2.0 library in order to handle JUMBF data. In the following subclauses two different types of applications are considered: a command line interface application and a web application. Both applications are built using the Spring Boot framework.[4]

B.3.2 Command line interface

The first software is a Spring Boot application that implements a command line interface to handle JUMBF data. In particular, it supports the generation of JUMBF files by providing csv files that describe the structure of the JUMBF data to be generated either as standalone files or embedded in JPEG encoded codestreams. In parallel, it provides the command to parse images and/or ".jumbf" files and create a report summarizing the contents of JUMBF Box that is identified.