ISO/IEC JTC 1/SC 29/WG1M96015
96th meeting – Online – July 25-29, 2022

ISO/IEC JTC 1/SC 29/WG 1
(ITU-T SG16)

Coding of Still Pictures

JBIG	JPEG
Joint Bi-level Image Experts Group	Joint Photographic Experts Group

TITLE: UPC proposal for JPEG Systems' Privacy & Security (ISO/IEC 19566-4) Reference Software v2

SOURCE: Nikolaos Fotos, Jaime Delgado (DMAG/IMP-UPC)
Distributed Multimedia Applications Group (DMAG) /
Information Modeling and Processing Research Group (IMP)
Universitat Politècnica de Catalunya – UPC BarcelonaTECH
jaime.delgado@upc.edu, nikolaos.fotos@estudiantat.upc.edu

PROJECT: ISO/IEC 19566-4
(JPEG Systems - Part 4: Privacy & Security)

STATUS: Proposal

REQUESTED ACTION: Input contribution

DISTRIBUTION: WG1

UPC proposal for JPEG Systems' Privacy & Security (ISO/IEC 19566-4) Reference Software: An application of the proposed JUMBF Reference Software v2

Nikolaos Fotos, Jaime Delgado

UPC BarcelonaTECH

1. Introduction

Document ISO/IEC JTC1 SC29/WG1/M96014: “UPC proposal for JUMBF (ISO/IEC 19566-5) Ed2 Reference Software” presents a proposal for Reference Software associated to ISO/IEC 19566-5; i.e. JPEG Systems - Part 5: JPEG universal metadata box format (JUMBF). One of the main characteristics of the proposed JUMBF reference software is the ability to support additional JUMBF boxes defined in other JPEG standards.

This document describes how we could extend the functionality of the proposed JUMBF reference software in order to support the JUMBF boxes defined in ISO/IEC 19566-4, i.e. JPEG Systems - Part 4: Privacy and Security. The new boxes that we present in the following sections are the Protection Box and the Replacement Box.

The complete mipams-jumbf project is implemented in Java. In this document we present an extension of the initial jumbf-privsec-1.0 library for JUMBF supporting the additional Protection and Replacement Boxes defined in the JPEG Systems Part 4 standard. The first version of this library is in document ISO/IEC JTC1 SC29/WG1/M95049.

The document assumes the knowledge of the definitions and terms presented in the companion document regarding the proposal of the JUMBF Ed2 reference software (document ISO/IEC JTC1 SC29/WG1/M96014, already mentioned).

Section 2 provides the class definitions that need to be implemented in order to support the Protection box structure. Next, Section 3 presents the respective classes for the Replacement box. Finally, Section 4 presents two examples on how we can use the demo application defined in the first document to generate a Protection and a Replacement box.

The complete software is available at https:www.github.com/nickft/mipams-jumbf, while the specific software for Part 4 described here is in https://github.com/nickft/mipams-jumbf/tree/main/privsec

2. Protection Box

As per ISO/IEC 19566-4, a Protection box is defined as a JUMBF box with a new Content Type UUID consisting of two Content Boxes, namely a Protection Description box and a

Binary Data box. Consequently, we need to extend jumbf-core library in order to support the new Protection Description box Entity and Service classes and define the new Protection Content Type box.

Regarding the Protection Description box, we need to, first, define its Entity class which specifies the fields listed in this box. As explained in the first input document (WG1/M95049) for the proposal, each Box is wrapped with the ISOBMFF headers. Thus, the new ProtectionDescriptionBox class needs to extend the BmffBox class as shown in Figure 1.

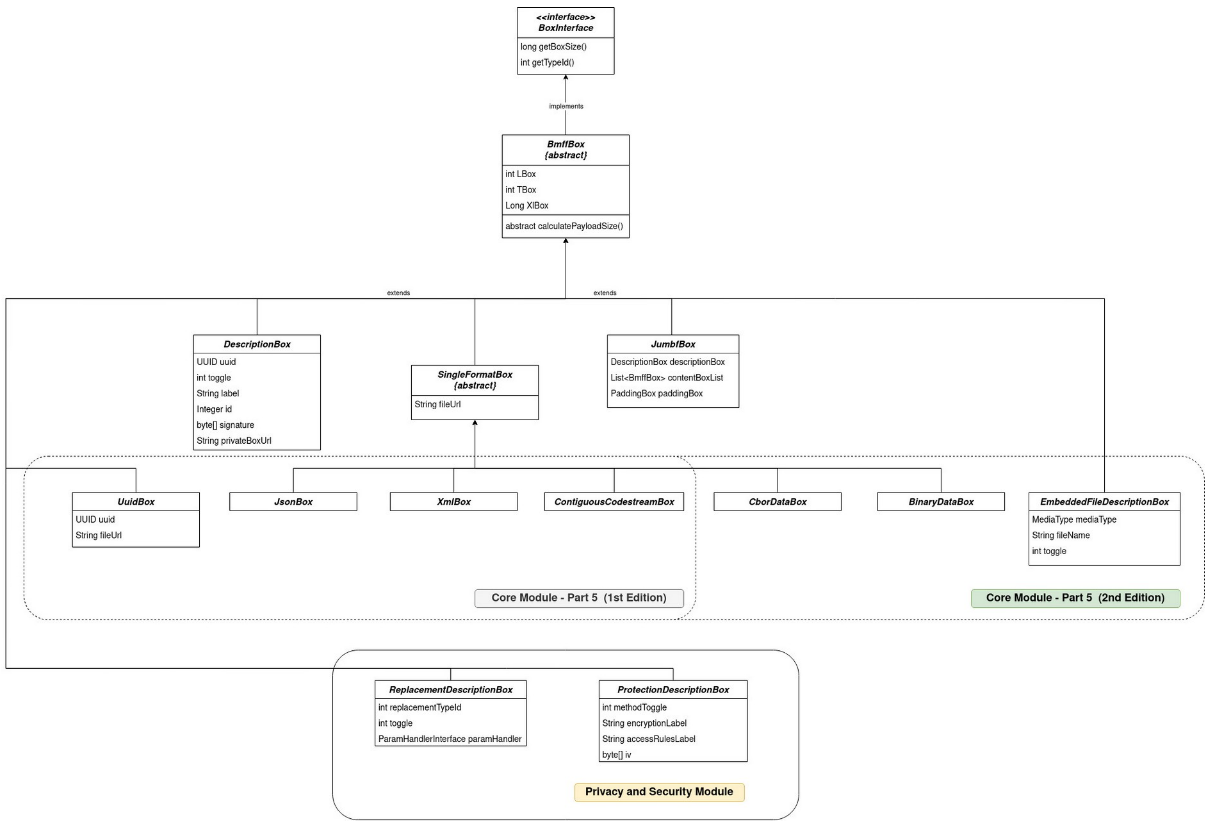
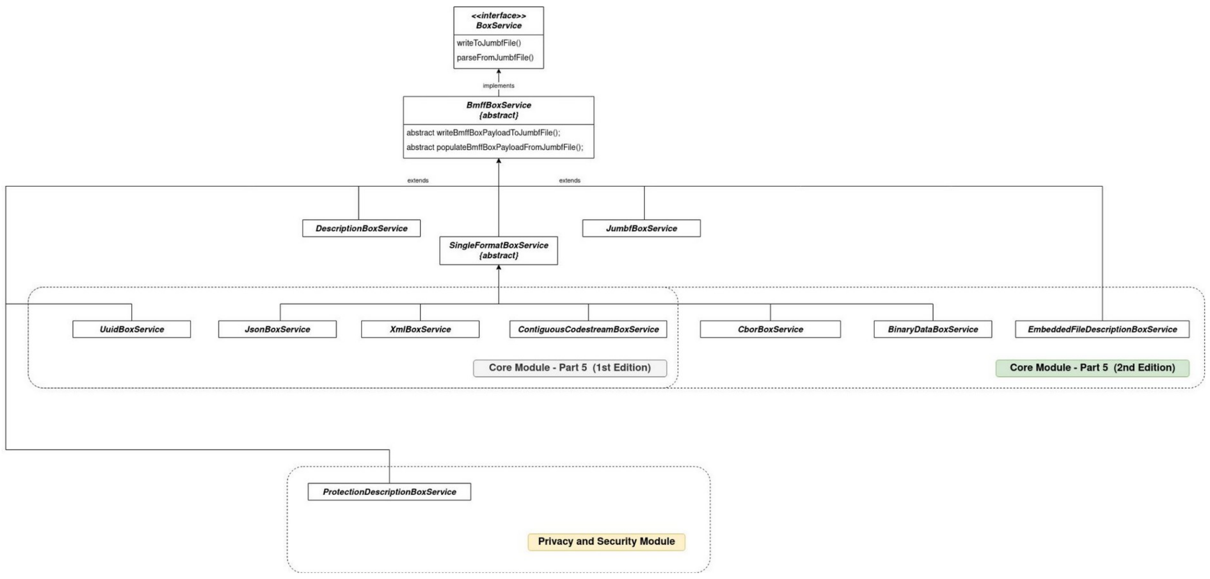


Figure 1: Extending Entity class hierarchy with Protection Description box

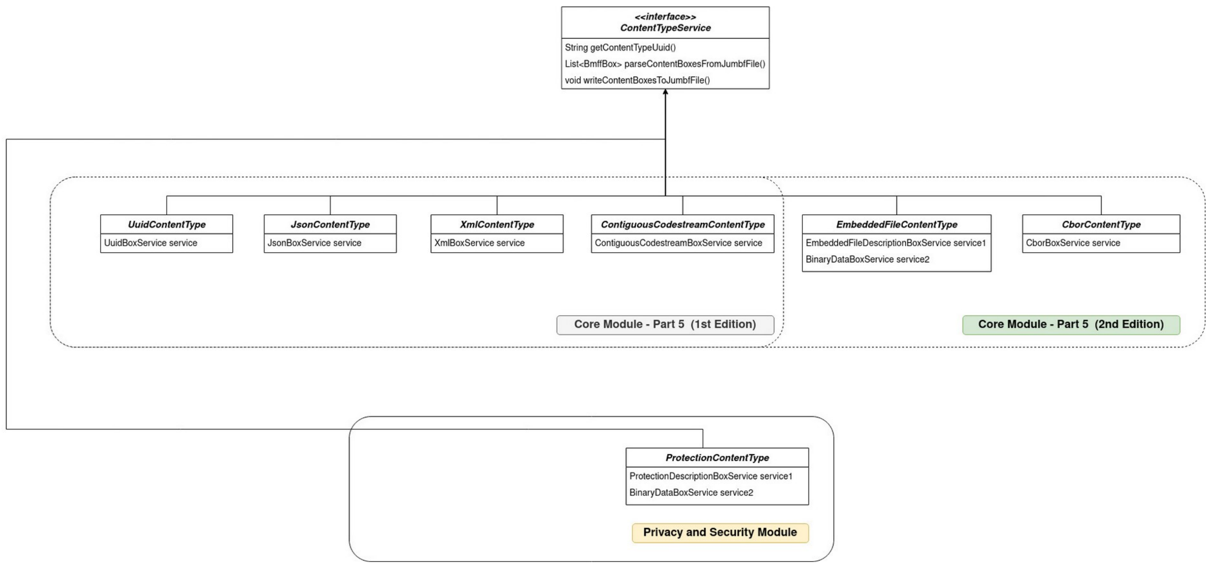
Next, we need to define the respective Service class that parsers/generates the fields that reside in a Protection Description box. Specifically, in Figure 2, we demonstrate how ProtectionDescriptionService class extends the functionality of BmffBoxService class.



<<interface>>
BoxService
writeToJumbfFile()
parseFromJumbfFile()
implements
BmffBoxService
{abstract}
abstract writeBmffBoxPayloadToJumbfFile();
abstract populateBmffBoxPayloadFromJumbfFile();
extends
extends
DescriptionBoxService
JumbfBoxService
SingleFormatBoxService
{abstract}
UuidBoxService
JsonBoxService
XmlBoxService
ContiguousCodestreamBoxService
CborBoxService
BinaryDataBoxService
EmbeddedFileDescriptionBoxService
Core Module - Part 5 (1st Edition)
Core Module - Part 5 (2nd Edition)
ProtectionDescriptionBoxService
Privacy and Security Module

Figure 2: Extending Service class hierarchy with Protection Description box

At this point we have all the necessary boxes in order to define the new Protection Content Type. As specified in the first input document, a ContentType class in jumbf-core library provides the means to parse/generate the content boxes of a JUMBF box. Depending on the Content Type UUID that resides in the Description box, the JumbfBoxService class decides the proper ContentType class to handle the JUMBF Box Content Boxes. The definition of the ProtectionContentType class is depicted in Figure 3.



<<interface>>
ContentTypeService
String getContentTypeUuid()
List<BmffBox> parseContentBoxesFromJumbfFile()
void writeContentBoxesToJumbfFile()
UuidContentType
UuidBoxService service
JsonContentType
JsonBoxService service
XmlContentType
XmlBoxService service
ContiguousCodestreamContentType
ContiguousCodestreamBoxService service
EmbeddedFileContentType
EmbeddedFileDescriptionBoxService service1
BinaryDataBoxService service2
CborContentType
CborBoxService service
Core Module - Part 5 (1st Edition)
Core Module - Part 5 (2nd Edition)
ProtectionContentType
ProtectionDescriptionBoxService service1
BinaryDataBoxService service2
Privacy and Security Module

Figure 3: Extending Content Type class hierarchy with Protection Content type

Notice that in the ProtectionContentType class there are two services that are defined. One for handling the Protection Description box and one for the Binary Data box.

3. Replacement Box

This section focuses on the Replacement Content type JUMBF box definition supporting all the different types of replacement defined in ISO/IEC 19566-4.

In general, the Content Box set of a Replacement Content type JUMBF box contains one Replacement Description Box and one or more Replacement Data Boxes, depending on the type of replacement. Specifically, there are four different types of replacement:

- **Box Replacement**: Replacing a box referenced by either an offset or a label with a list of one or more boxes specified in the Replacement Data Box section.
- **APP Replacement**: Replacing an APP marker segment that is referenced using the offset in the file with the contents (i.e. one or more app segments) of exactly one Replacement Data box which is a Binary Data Box.
- **ROI Replacement**: Replacing a region of interest (ROI) - as specified by the corresponding offset - in the parent image with the content of exactly one Replacement Data box which is a Contiguous Codestream box.
- **File Replacement**: Replacing the entire file where this Replacement box resides with the content of exactly one Replacement Data box which is a Contiguous Codestream box.

First of all, a new box is defined, namely the Replacement Description Box. Hence, a new class ReplacementDescriptionBox is created extending the BmffBox class as shown in Figure 4.

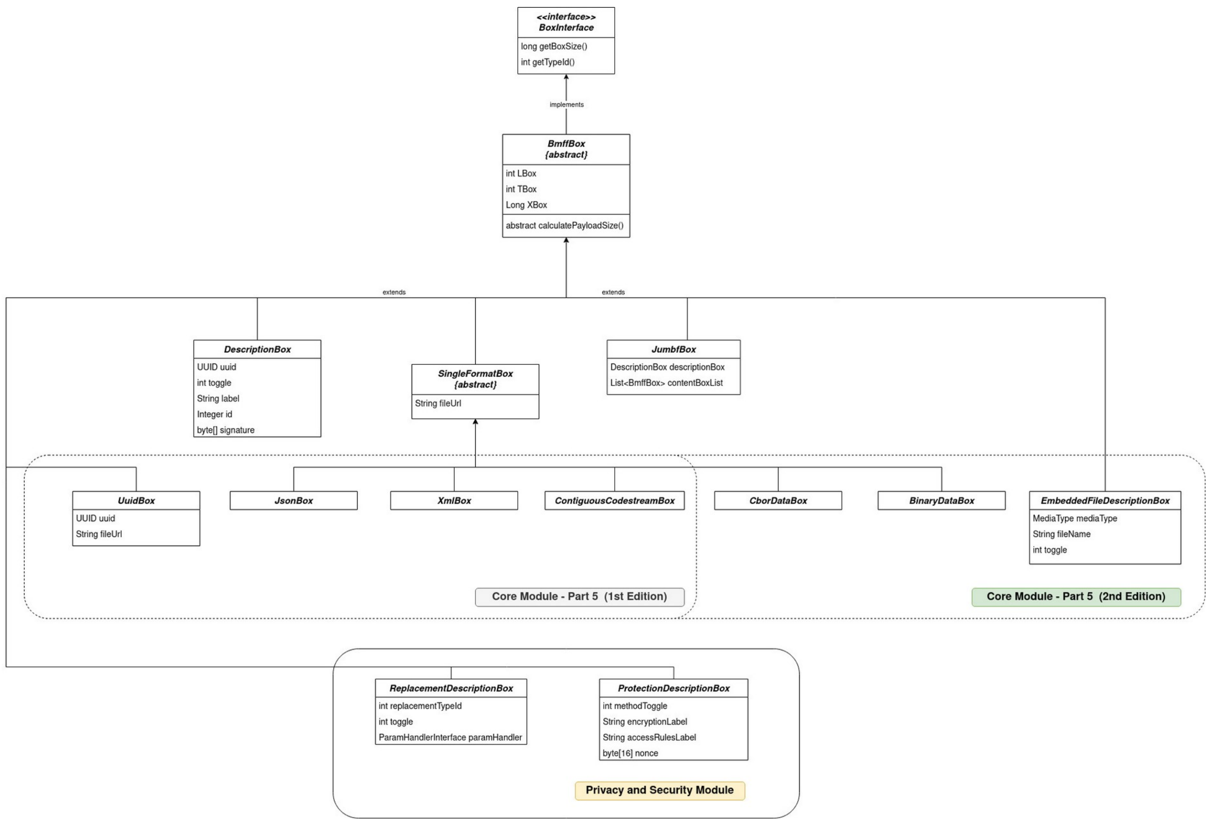


Figure 4: Extending Entity class hierarchy with Replacement Description box

Each replacement type defines a different set of parameters. However, since the resulting Replacement Content type is the same for all replacement types (i.e. single Content Type UUID), we decided to implement a single ReplacementDescriptionBox class and define different parameter handler classes handling the parameters of each replacement type.

Thus, as shown at the bottom of Figure 4, we have included a field of type ParamHandlerInterface in the Replacement Description Box Entity class. These ParamHandlerInterface classes - apart from specifying the exact parameters of each replacement type - provide the functionality to parse and generate the specific parameter fields to bytes in order to be later included in the Replacement Description Box JUMBF structure. The class inheritance of the parameter handling classes is depicted in Figure 5.

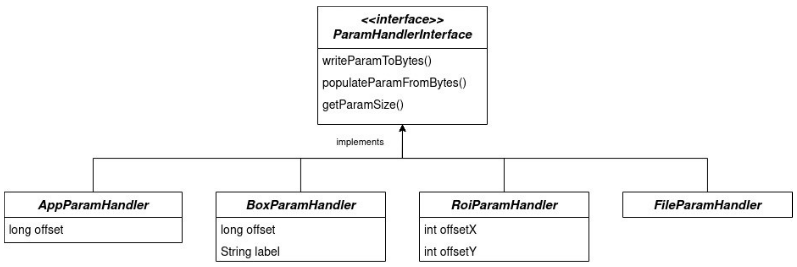


Figure 5: Replacement Description box parameter handling class inheritance

Subsequently, it is required that we specify the methods that parse and generate a Replacement Description box to a JUMBF structure. Alternatively, we need to define the ReplacementDescriptionBoxService class as shown in Figure 6.

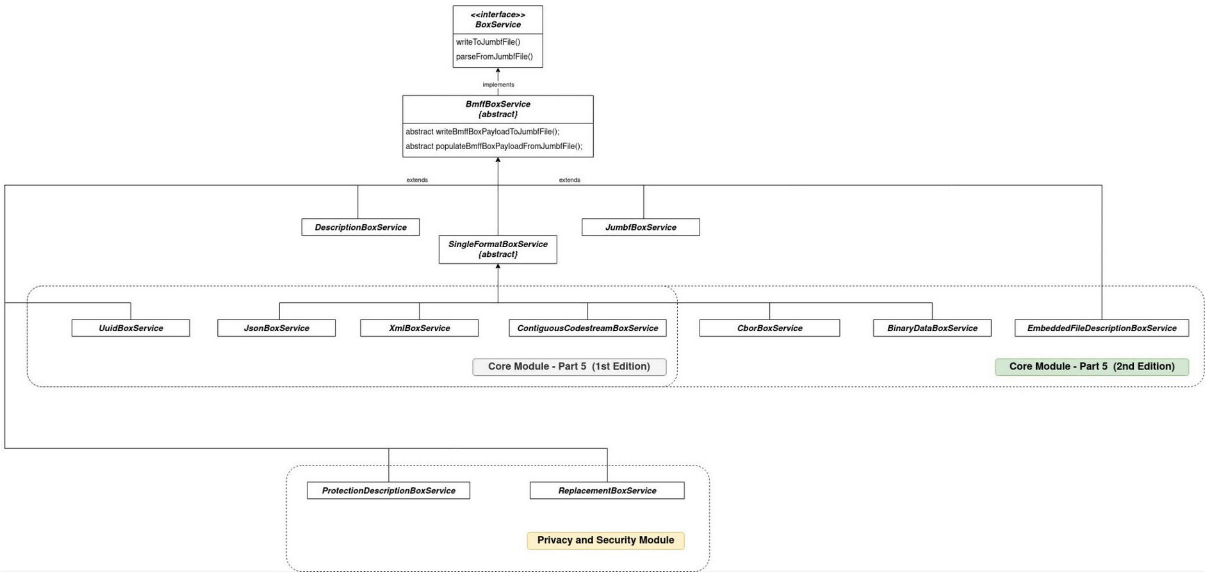

<<interface>>
BoxService
writeToJumbfFile()
parseFromJumbfFile()
implements
BmffBoxService
{abstract}
abstract writeBmffBoxPayloadToJumbfFile();
abstract populateBmffBoxPayloadFromJumbfFile();
extends
extends
DescriptionBoxService
SingleFormatBoxService
{abstract}
JumbfBoxService
UuidBoxService
JsonBoxService
XmlBoxService
ContiguousCodestreamBoxService
CborBoxService
BinaryDataBoxService
EmbeddedFileDescriptionBoxService
Core Module - Part 5 (1st Edition)
Core Module - Part 5 (2nd Edition)
ProtectionDescriptionBoxService
ReplacementBoxService
Privacy and Security Module

Figure 6: Extending Service class hierarchy with Replacement Description box

At this point we have all the necessary boxes in order to define the new Replacement Content type, namely ReplacementContentType class, the definition of which is depicted in Figure 7.

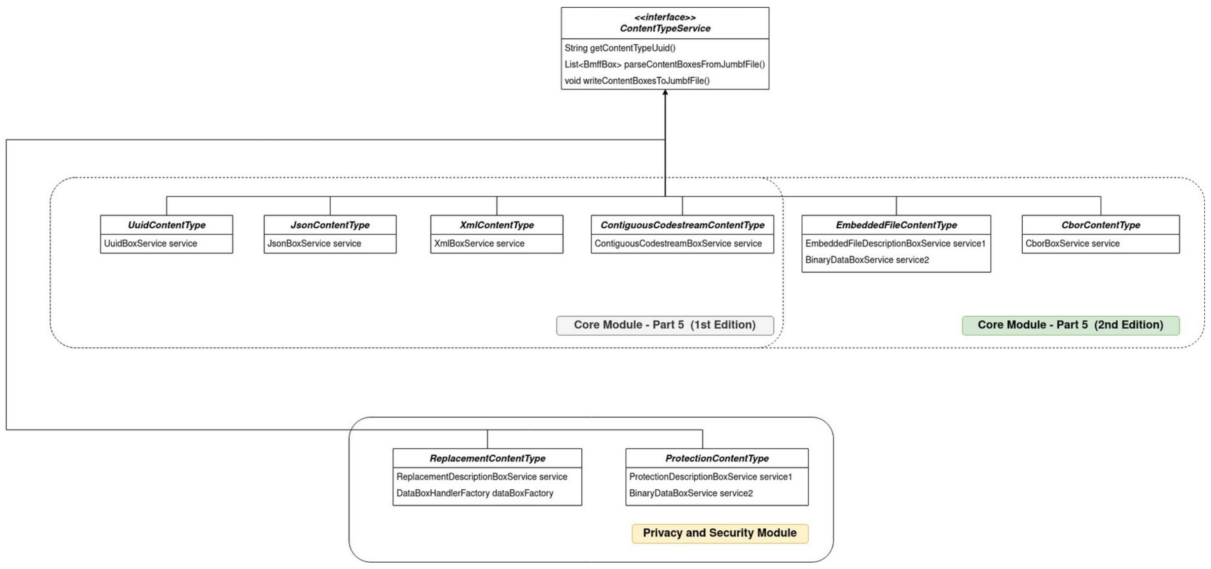

<<interface>>
ContentTypeService
String getContentTypeUuid()
List<BmffBox> parseContentBoxesFromJumbfFile()
void writeContentBoxesToJumbfFile()
UuidContentType
UuidBoxService service
JsonContentType
JsonBoxService service
XmlContentType
XmlBoxService service
ContiguousCodestreamContentType
ContiguousCodestreamBoxService service
EmbeddedFileContentType
EmbeddedFileDescriptionBoxService service1
BinaryDataBoxService service2
CborContentType
CborBoxService service
Core Module - Part 5 (1st Edition)
Core Module - Part 5 (2nd Edition)
ReplacementContentType
ReplacementDescriptionBoxService service
DataBoxHandlerFactory dataBoxFactory
ProtectionContentType
ProtectionDescriptionBoxService service1
BinaryDataBoxService service2
Privacy and Security Module

Figure 7: Extending Content Type class hierarchy with Replacement Content type

Notice that, as in the case of Protection Content type, there are two services residing in the ReplacementContentType class. The first one, is responsible for handling the Replacement Description box while the other one, namely DataBoxHandlerFactory class provides the means to write the corresponding Replacement Data Boxes according to the replacement type.

Specifically, we have defined a DataBoxHandler interface which, as described in Figure 8, provides the means to parse/generate the Replacement Data Boxes of each replacement type. For the selection of the correct DataBoxHandler, a new DataBoxHandlerFactory class is defined that acts based on the replacementTypeId field provided by the Replacement Description Box.

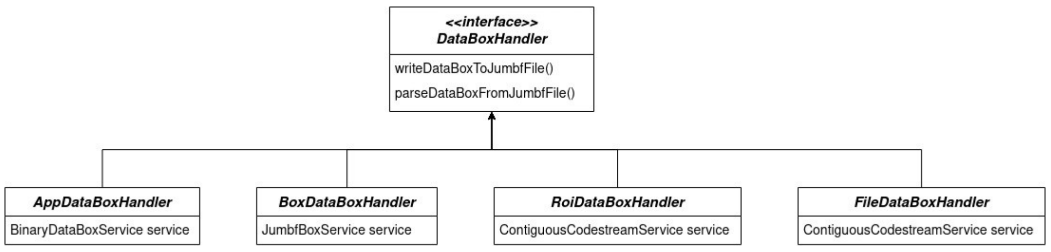

<<interface>>
DataBoxHandler
writeDataBoxToJumbfFile()
parseDataBoxFromJumbfFile()
AppDataBoxHandler
BinaryDataBoxService service
BoxDataBoxHandler
JumbfBoxService service
RoiDataBoxHandler
ContiguousCodestreamService service
FileDataBoxHandler
ContiguousCodestreamService service

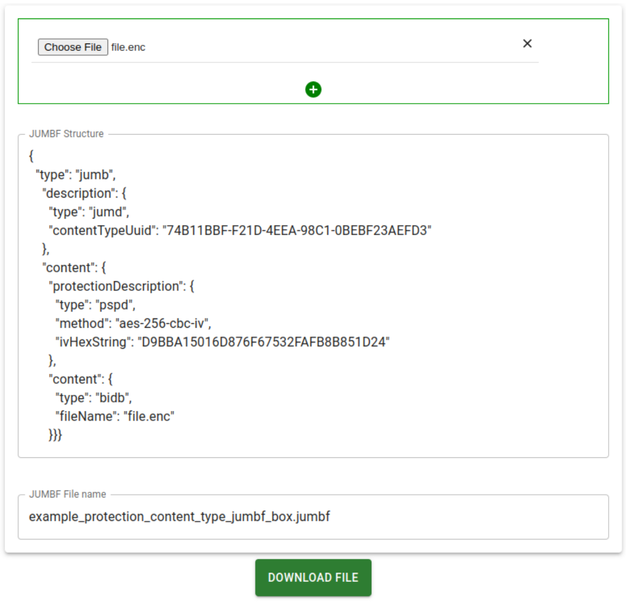
Figure 8: Replacement Data box handling class inheritance

Finally, notice that in each DataBoxHandler class there is a service field assigned. As we saw in the introduction of this section, the supported Replacement Data Boxes consist of boxes that are already defined in scope of the input document for JUMBF Reference Software (WG1/M96014). Thus, the services to parse/generate the Replacement Data Boxes have already been defined.

4. Demo application

In this section we use the example application implemented in the scope of the UPC proposal for JUMBF (ISO/IEC 19566-5) Ed2 Reference Software” (WG1/M96014) in order to parse and generate two JUMBF boxes: one with Protection Content type and one with Replacement Content type.

First, we examine the case of the Protection Content type JUMBF box. For this example we assume that we have already encrypted some piece of information using AES-256-CBC mode with a specific initial value (IV) and the ciphertext is located in a file called “file.enc”. In Figure 9, we can view the steps that the user must perform in order to generate a Protection Content type JUMBF box expressing the aforementioned example.

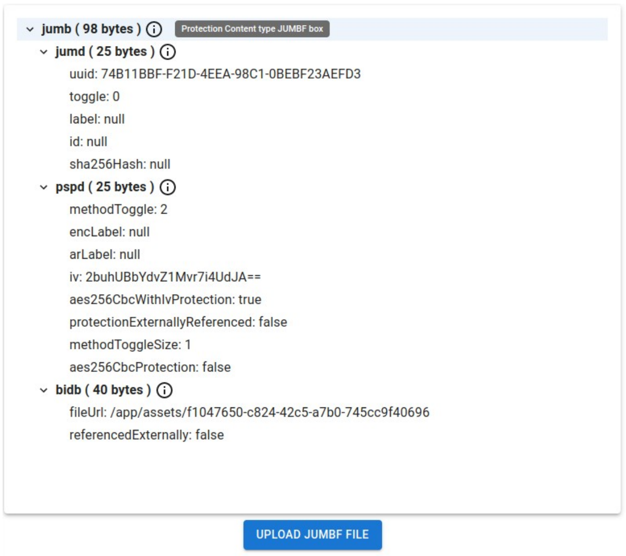


```
{
 "type": "jumb",
  "description": {
   "type": "jumd",
   "contentTypeUuid": "74B11BBF-F21D-4EEA-98C1-0BEBF23AEFD3"
  },
  "content": {
   "protectionDescription": {
    "type": "pspd",
    "method": "aes-256-cbc-iv",
    "ivHexString": "D9BBA15016D876F67532FAFB8B851D24"
   },
   "content": {
    "type": "bidb",
    "fileName": "file.enc"
   }}}
```

example_protection_content_type_jumbf_box.jumbf

Figure 9: Generating a Protection Content type JUMBF box

Once the user downloads the file and tries to parse it to examine its structure from the GUI. The respective JUMBF structure is depicted in Figure 10.



```
jumb ( 98 bytes )
  jumd ( 25 bytes )
    uuid: 74B11BBF-F21D-4EEA-98C1-0BEBF23AEFD3
    toggle: 0
    label: null
    id: null
    sha256Hash: null
  pspd ( 25 bytes )
    methodToggle: 2
    encLabel: null
    arLabel: null
    iv: 2buhUBbYdvZ1Mvr7i4UdJA==
    aes256CbcWithIvProtection: true
    protectionExternallyReferenced: false
    methodToggleSize: 1
    aes256CbcProtection: false
  bidb ( 40 bytes )
    fileUrl: /app/assets/f1047650-c824-42c5-a7b0-745cc9f40696
    referencedExternally: false
```

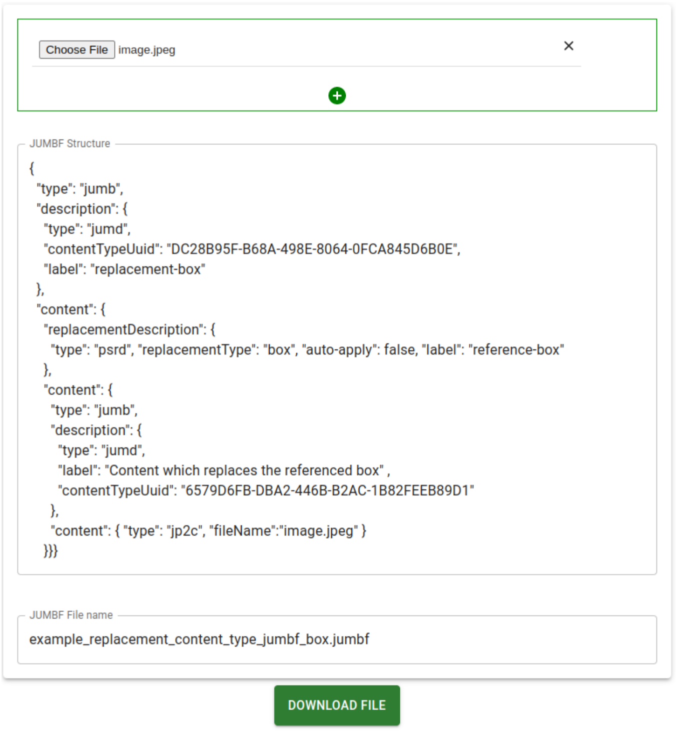
Figure 10: Parsing a JUMBF file

From the GUI parser the user can examine the fields that each Box Entity contains in the mipams-jumbf project. Additionally, the user can examine the size of each Box as well as descriptive information about each node by hovering over the “Information” icon. In the case of a JUMBF box this description explains the Content Type of the JUMBF Box while in any other case the description of the ISOBMFF TBox type is provided.

Note: Upon parsing the JUMBF file, jumbf-core-2.0 library extracts the embedded information (in this case the ciphertext byte stream which was imported from the “file.enc” file) and stores it in a separate file with the corresponding format and a random name.

In the second example we generate and parse a Replacement Content type JUMBF box. For this example we assume that the user wants to express the Box replacement of a JUMBF Box with the label “reference-box” with a Contiguous Codestream Content type JUMBF box containing the image located in the “image.jpeg” file.

The way to express this Replacement Content type JUMBF box is depicted in Figure 11.



```
{
 "type": "jumb",
 "description": {
  "type": "jumd",
  "contentTypeUuid": "DC28B95F-B68A-498E-8064-0FCA845D6B0E",
  "label": "replacement-box"
 },
 "content": {
  "replacementDescription": {
   "type": "psrd", "replacementType": "box", "auto-apply": false, "label": "reference-box"
  },
  "content": {
   "type": "jumb",
   "description": {
    "type": "jumd",
    "label": "Content which replaces the referenced box" ,
    "contentTypeUuid": "6579D6FB-DBA2-446B-B2AC-1B82FEEB89D1"
   },
   "content": { "type": "jp2c", "fileName":"image.jpeg" }
  }}}
```

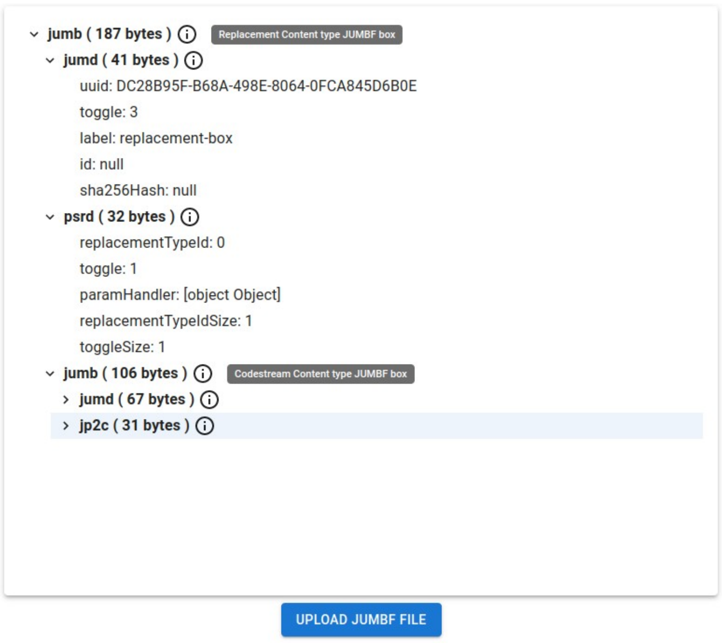
example_replacement_content_type_jumbf_box.jumbf

Figure 11: Generating a Replacement Content type JUMBF box

For more information about the syntax to generate a JUMBF file, visit the Github repository specified in the introduction of the document.

Note: The creation of the referenced box (i.e. the JUMBF Box that has the label “reference-box”) is out of scope of this example. In fact, it is out of scope of the jumbf-core-2.0 library to validate the context of the JUMBF Boxes that are being parsed or created. It is assumed that the application that uses jumbf-core-2.0 and jumbf-privsec-1.0 libraries handles the relationships between JUMBF Boxes.

Finally, in Figure 12 we can see the results of parsing the JUMBF file that the user has just generated.



```
⌄ jumb ( 187 bytes ) ⓘ  Replacement Content type JUMBF box
  ⌄ jumd ( 41 bytes ) ⓘ
      uuid: DC28B95F-B68A-498E-8064-0FCA845D6B0E
      toggle: 3
      label: replacement-box
      id: null
      sha256Hash: null
  ⌄ psrd ( 32 bytes ) ⓘ
      replacementTypeId: 0
      toggle: 1
      paramHandler: [object Object]
      replacementTypeIdSize: 1
      toggleSize: 1
  ⌄ jumb ( 106 bytes ) ⓘ  Codestream Content type JUMBF box
    › jumd ( 67 bytes ) ⓘ
    › jp2c ( 31 bytes ) ⓘ
```

Figure 12: Parsing a JUMBF file

What is worth mentioning in this example is that this Replacement Content type JUMBF box that we see in Figure 12 is a superbox as it contains another JUMBF box as part of its Content Boxes.

5. Conclusions

This document presents the means to extend the software introduced in document ISO/IEC JTC1 SC29/WG1/M96014: "UPC proposal for JUMBF (ISO/IEC 19566-5) Ed2 Reference Software". Specifically, we have examined how to define the Entity and Service classes for the Protection Box. Subsequently, we investigated how to define the Replacement Box as well as how to handle the different replacement types.

Finally, we demonstrate how to use the example application presented in the first input document in order to generate and parse Protection and Replacement JUMBF boxes through the graphical user interface.

The software presented here could be the basis for JPEG Systems' Privacy and Security (Part 4) Reference Software, to be potentially included in the future Part 10 of JPEG Systems.

Acknowledgements

The work presented in this contribution has been partially supported by the Spanish Government under the project GenClinLab-Sec (PID2020-114394RB-C31, funded by MCIN/AEI/10.13039/501100011033) and by the Generalitat de Catalunya (2017 SGR 174).