European Commission

JRC SCIENTIFIC INFORMATION SYSTEMS AND DATABASES

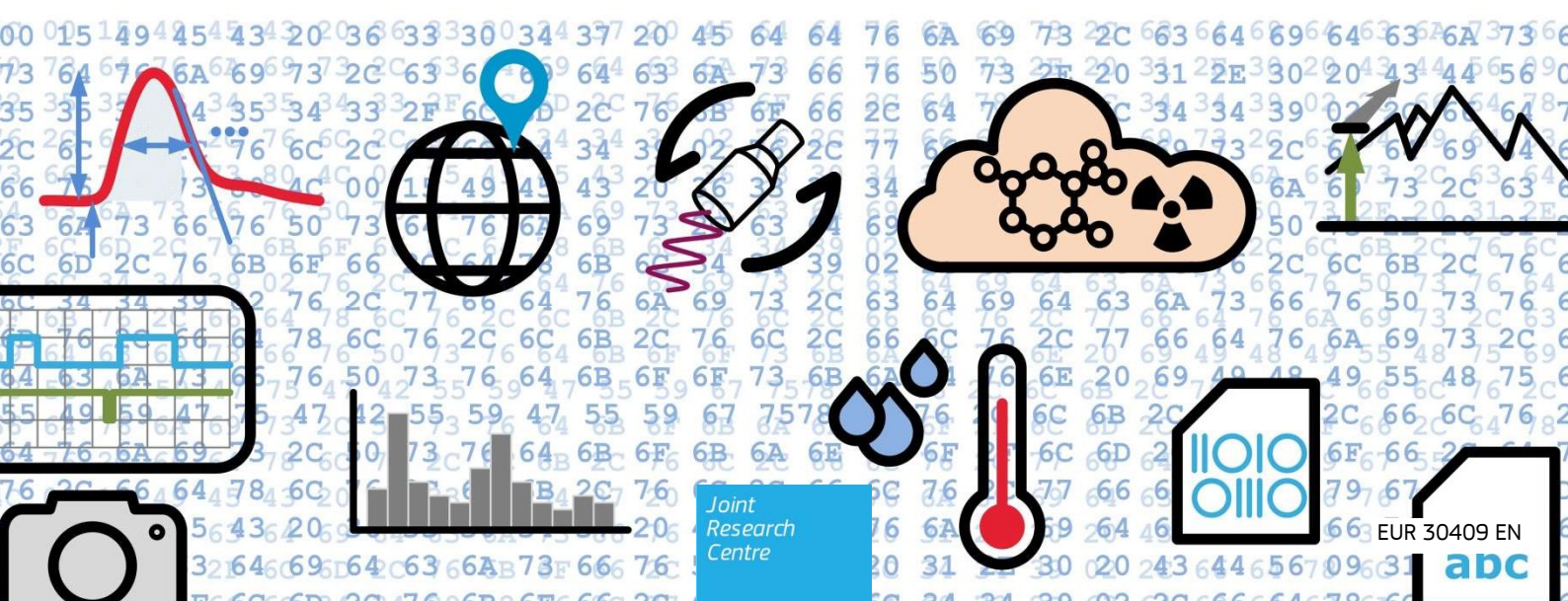
An open-source solution for encoding and decoding IEC 63047:2018 data

User manual for an IEC 63047 codec

Paepen, J.

Lutter, G.

2021



EUR 30409 EN

Contact information
Name: Jan Paepen
Address: JRC-Geel, Retieseweg 111, B-2440 Geel, Belgium
Email: jan.paepen@ec.europa.eu
Tel.: +32 14 571329

EU Science Hub
https://ec.europa.eu/jrc

JRC121882

EUR 30409

PDF ISBN 978-92-76-23523-1 ISSN 1831-9424 doi:10.2760/644570

Luxembourg: Publications Office of the European Union, 2021

How to cite this report: Paepen, J. and Lutter, G., An open-source solution for encoding and decoding IEC 63047:2018 data, EUR 30409 EN, Publications Office of the European Union, Luxembourg, 2021, ISBN 978-92-76-23523-1, doi:10.2760/644570, JRC121882.

Contents

Abstract

IEC 63047 is a standard specifying the format of list-mode data acquired by nuclear data acquisition instruments. For performance reasons, it is a binary format. It is defined using the internationally standardised syntax notation ASN.1, for which commercially available and open-source compilers exist that convert the ASN.1 definition of IEC 63047 into software code to encode (write) and decode (read) IEC 63047 data.

This report provides an open-source solution, built on the open-source code asn1c from GitHub, and includes demo source code developed by the JRC. The report is aimed at developers of software who need to write or read IEC 63047 data. The demo code uses a limited number of IEC 63047 data types and is intended to serve as a basis for developing more elaborate code for the user's specific application.

1 Introduction

On 11 October 2018, the International Electrotechnical Commission (IEC) published a new International Standard, IEC 63047 ([1]), specifying a data format for list-mode digital data acquisition used in radiation detection and measurement. The European Commission's Joint Research Centre (DG JRC) led the development of the standard, in support of the Commission's Directorate General for Migration and Home Affairs (DG HOME) in the frame of Commission mandate M/487 and with input from industrial stakeholders and Member States organisations.

IEC 63047 is a binary format defined using ASN.1 (Abstract Syntax Notation One), an internationally standardised syntax notation for specifying data formats, defined in ISO/IEC 8824-1, ITU-T X.680 ([2]). The ASN.1 syntax notation is complemented with standardised encoding rules, which determine the bit pattern corresponding to the data values. The IEC 63047 format uses the canonical octet encoding rule (C-OER), defined in ISO/IEC 8825-7, ITU-T X.696 ([3]).

IEC 63047 enables encryption and authentication of the data. The format supports various types of timestamped data and can be used in a wide range of applications involving radiation detection and measurement. The standard includes data types to represent the geolocation and measurement results of any kind of sensor, which allows its use in mobile applications, e.g. CBRNE sensor networks.

To lower the threshold and facilitate the use of the standard, this document provides an open-source solution for writing (encoding) and reading (decoding) IEC 63047 data. The solution uses the open-source software `asn1c`, developed by Lev Walkin and contributors, and available on GitHub ([4]). This document explains how to use `asn1c` to generate an IEC 63047 codec and provides guidance for implementing the standard in radiation detection systems.

Box 1. Note about source code

The source code in the annexes of this manual is also included as an attachment included in the pdf file of this document.

2 Scope

2.1 General

This document provides instructions on how to use the open-source code `asn1c` developed by Lev Walkin and contributors ([4]) to create a codec for IEC 63047. The codec includes software routines for encoding (writing) and decoding (reading) binary data, formatted according to the International Standard IEC 63047:2018. Demo code developed by the JRC explains the use of the codec. This document applies to the edition of the standard and the version of `asn1c` specified in Table 1.

Table 1. Summary of scope.

Standard and edition	IEC 63047, Edition 1.0, 2018-10 Nuclear instrumentation – Data format for list mode digital data acquisition used in radiation detection and measurements
Version of `asn1c` used to develop the codec	Commit 00fa516 of 07 July 2020

2.2 Prerequisites

The user should refer to IEC 63047 for an explanation on how to use the standard. Also, the user will need, as a text file, the ASN.1 specification of the data format, taken from Annex A of IEC 63047. This file is provided together with the standard.

2.3 Targeted audience

This user manual is aimed at software developers and system integrators that need to encode or decode binary IEC 63047 messages. Encoding of IEC 63047 data is usually performed by the data acquisition software associated with digital data acquisition instrumentation while decoding is performed by the software that reads, analyses or converts IEC 63047 formatted data.

2.4 Supported platform and operating system

`Asn1c` is developed for the Linux operating system, but runs on Windows systems or other operating systems, although some modifications may be necessary. `Asn1c` and the demo code were successfully tested on the processors and operating systems shown in Table 2 by following the workflow in section 4.

Table 2. Platforms on which the workflow in section 4 was successfully tested.

Compile and execute `asn1c`	**Compile and execute the demo code**
gcc Red Hat 4.8.5 compiler, CentOS Linux release 7.4, x86-64	Same
gcc Red Hat 4.8.5 compiler, CentOS Linux release 7.4, x86-64	gcc 6.3 compiler, Raspbian 9.1 (Debian-based), running on a Raspberry Pi 3 Model B, quad-core ARM Cortex A53 (ARSMv8)
Cygwin 2.9.0 64-bit, running on Windows 10	Same
Linux Ubuntu subsystem running on Windows 10, after installing the Fall Creators Update	Same
Microsoft Windows 10	MinGW with GCC 8.2.0

2.5 Disclaimer

The demo code is provided as is and only supports a subset of the data types defined in IEC 63047. The demo code is not intended to be complete, nor free of errors. Refer to the section 3 for the license conditions.

As an alternative to the open-source software `asn1c`, commercial software is available on the market to generate source code containing data structures and routines for encoding and decoding data specified in a format using the ASN.1 syntax of IEC 63047, or any other data format specified using the ASN.1 standards. These commercial so-called 'ASN.1 compilers' support various programming languages and computer platforms.

3 Licenses

It is imperative that users of the open-source code `asn1c` and the demo code consult the following licenses and adhere to the conditions therein.

3.1 Asn1c

The codec is built on the open-source code `asn1c` which is available on GitHub under the following BSD license:

Box 2. `Asn1c` copyright

3.2 Demo code

The demo code developed by the JRC is available under the EUPL v.1.2 license [5],[6], which is compatible with the `asn1c` license.

4 How to use this manual – ASN.1 workflow

Figure 1 presents the ASN.1 workflow, applied to the standard IEC 63047. Following the steps below to build a codec application:

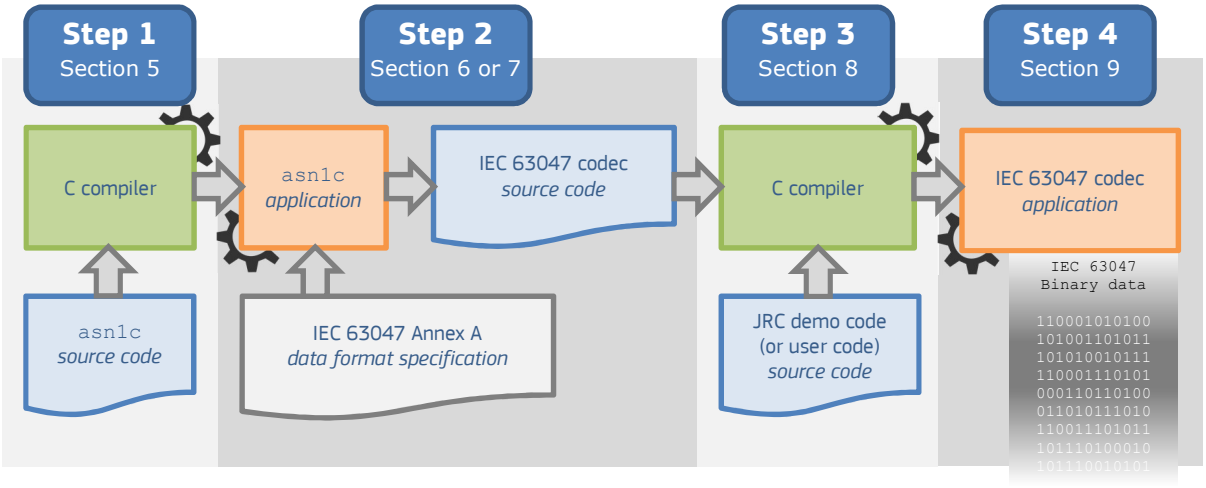
— Step 1: obtain the `asn1c` source code, compile and install the application (see section 5);

— Step 2: Use `asn1c` to generate C source code (.h and .c files) from the data format specification in Annex A of IEC 63047. (A text file with the ASN.1 specification is provided when obtaining the standard.) The C source code defines native C structures, which correspond to the IEC 63047 ASN.1 syntax, and provides encoding and decoding routines that convert the C values to the bit patterns and back, as defined by the C-OER encoding rule. Section 6 explains this step using a script, which is explained in detail in section 7.

— Step 3: Compile and link the JRC demo code (or user code) with the codec source code generated in the previous step (see section 8). The JRC demo code uses the C structures and encoding/decoding routines generated by asn1c to demonstrate how to encode and decode IEC 63047 data.

— Step 4: Start using the IEC 63047 codec application.

Figure 1. ASN.1 workflow.



5 Obtaining, compiling and installing asn1c

5.1 Obtaining asn1c

Download asn1c from GitHub (https://github.com/vlm/asn1c) and extract the file to a folder with name asn1c-master, for example in the home folder of the user.

```
[…]$ unzip asn1c-master.zip
[…]$ cd asn1c-master
```

5.2 Verification of the dependencies

The dependencies of asn1c are documented in the file REQUIREMENTS.md in the asn1c-master folder. Use the --version option to verify if the version of the installed packages on your machine corresponds with the requirements. For example:

```
[…]$ gcc --version
```

Note that the version of some of the components is linked to the version of the operating system. When the versions do not correspond with the required one, compilation may nevertheless still work.

5.3 Compiling and installing asn1c

Follow the instructions in INSTALL.md to configure (with default settings), build and install asn1c. Open a terminal window, browse to the asn1c-master folder and execute the instructions. (If you choose to install in the default folder, you have to be super-user. Return to your normal user with exit.) The commands are the following:

```
[…]$ test -f configure || autoreconf -iv
[…]$ ./configure
[…]$ make
[…]$ su
[…]# make install
[…]# exit
```

The end of INSTALL.md contains references to user guides for asn1c. More information can be found on http://lionet.info/asn1c/blog/.

5.4 Next steps

Follow the instructions in section 6 for generating the IEC 63047 codec source code automatically. Alternatively, follow the instructions in section 7 for generating the code manually. Section 7 also explains the changes that have to be applied on the ASN.1 specification of IEC 63047 before the codec source code can be generated by asn1c and some patches that have to be applied on the generated code.

6 Generating the IEC 63047 codec source code using a script

6.1 Actions performed by the script

The script `gen.sh` from Annex 1 generates the codec source code automatically. The script performs the actions explained further in section 7:

— It modifies the ASN.1 specification from Annex A of IEC 63047 (contained in the file `IEC63047.asn`) and makes it compatible with `asn1c`, without changing the data format itself. The new ASN.1 specification is saved with the name `IEC63047LMT.asn`.

— It runs the `asn1c` 'compiler' on `IEC63047LMT.asn` to generate the codec source code, supporting the C-OER encoding rule.

— It applies a patch on the generated code.

— It moves the generated code in the `.h` and `.c` files to subfolders `inc` and `src` respectively.

— It creates folders `tmp` and `bin`, in preparation of building the application.

6.2 Running the script

The ASN.1 specification in Annex A of IEC 63047 is provided by the IEC as a file with name `IEC63047.asn` together with the pdf of the standard.

Create a working folder and copy the file `IEC63047.asn` to it.

Copy the bash script `gen.sh` from the JRC demo code (or copy the text from Annex 1 into a file with the name `gen.sh`) and run in the working folder:

```
[…]$ ./gen.sh
```

6.3 Next steps

Be aware of some issues with `asn1c` related to decoding (see section 1017). When the codec source code is generated it can be compiled together with user code or the JRC demo code into an application that writes or reads IEC 63047 compliant binary data. Refer to section 8 for instructions on building the application.

7 Generating the IEC 63047 codec source code manually

7.1 Introduction

As an alternative to using the script (explained in section 6), use the following instructions to generate the codec source code manually.

7.2 Preparing the ASN.1 specification to make it compatible with `asn1c`

The ASN.1 specification in Annex A of IEC 63047 is provided by the IEC as a file with name `IEC63047.asn` together with the pdf of the standard.

Create a working folder as subfolder in `asn1-master`, for example with the name `IEC63047`, and copy the file `IEC63047.asn` to the working folder.

The element `date-time` of the `UTCDateTime` type defined in the standard uses the pre-defined ASN.1 type `TIME`, with some additional settings. This definition is currently not supported by `asn1c`. Running `asn1c` on the ASN.1 specification from the standard will result in the following error:

```
ASN.1 grammar parse error near IEC63047.asn:39 (token """): syntax error,
unexpected TOK_cstring, expecting ')'
Cannot parse "IEC63047.asn"
```

In order to make the ASN.1 specification in the file `IEC63047.asn` compatible with the version of `asn1c` from Table 1, the definition of the `date-time` element of the `UTCDateTime` type needs to be replaced.

Box 3. Important note related to the changes applied on the ASN.1 specification.

The change concerns only the syntax of the ASN.1 specification of IEC 63047; the data format itself remains unaltered. The change is merely the implementation of the C-OER encoding rule of the `TIME` type, according to ISO/IEC 8825-7 ([3]).

To apply the change, open the file `IEC63047.asn`, find the definition of the type `UTCDateTime` and locate the `date-time` element. The `date-time` element is currently defined via the `TIME` type. Remove the part `TIME (...)` and replace by `LMTime` (make sure to keep the comma) to yield:

```
date-time      LMTime,
```

Somewhere in the ASN.1 specification, the definition of the `LMTime` type needs to be added. Enter the following syntax at a location outside of any type definition:

```
LMTime ::= SEQUENCE {
  year      INTEGER,
  month     INTEGER (1..12),
  day       INTEGER (1..31),
  hour      INTEGER (0..24),
  min       INTEGER (0..59),
  sec       INTEGER (0..60)
}
```

Save the file with another filename, preferably `IEC63047LMT.asn` for consistency with this manual.

7.3 Performing a syntax check with `asn1c`

Use the following command to perform a syntax check on the ASN.1 specification in `IEC63047LMT.asn`:

```
 […]$ asn1c -EF IEC63047LMT.asn
```

`asn1c` will print the recognised syntax and if all goes well, no error messages will be generated.

7.4 Generate the codec with `asn1c`

Use the following command to generate the codec source code from `IEC63047LMT.asn`, implementing the C-OER encoding rule. The option `-fcompound-names` will avoid name collisions between data type

elements that have the same name. The option `-no-gen-example` tells `asn1c` not to generate its own example code.

```
[…]$ asn1c -no-gen-example -gen-OER -fcompound-names IEC63047LMT.asn
```

7.5 Patch on the files generated by asn1c

The version of `asn1c` from Table 1 generates a file `Footer.c`. This file results in the following errors when compiling:

```
Footer.c:193.4: error: 'asn_DEF_Member_7' undeclared here (not in a
function)
&asn_DEF_Member7,
Footer.c:230.4: error: 'asn_DEF_Member_9' undeclared here (not in a
function)
&asn_DEF_Member9,
...
```

Modify the file `Footer.c` and change both `&asn_DEF_Member7` and `&asn_DEF_Member9` to `&asn_DEF_NativeInteger`. In the file `REAL.c`, delete the text `CC_ATTR_NO_SANITIZE("float-cast-overflow")`.

7.6 Next steps

Be aware of some issues with `asn1c` related to decoding (see section 10). When the codec source code is generated it can be compiled together with user code or the JRC demo code into an application that writes or reads IEC 63047 compliant binary data. Refer to section 8 for instructions on building the application.

8 Building the codec application

After generating the IEC 63047 codec source, the JRC demo code or user code can be added. All code is then compiled and built into one single codec application, able to write or read IEC 63047 formatted data.

8.1 Obtaining the JRC demo code

The JRC demo code is attached to the pdf of this report as text files. Extract the files to the working folder. The demo code includes the files in Table 3. Source code and scripts are also included in the Annexes to this report.

Table 3. Files included in the JRC demo code.

File	Content
`gen.sh`	Bash script to generate the IEC 63047 codec source code (see section 6)

`lmdtest.c`
`Example1.h`
`Example1.c`
	JRC demo source code.

`Makefile`	The makefile to build the executable of the JRC demo code.
`test.sh`	Bash script to run the JRC demo code.
`In111.txt` `F18.txt`	Examples of histogram files that can be used by the JRC demo code.

8.2 Compiling the codec and the demo code

A `Makefile` is included to compile the codec and the demo code.

```
[…]$ make
```

The `Makefile` generates an executable with the name `lmdtest.x`. Section 9 explains how to use the executable.

9 Using the demo code

9.1 Operating modes

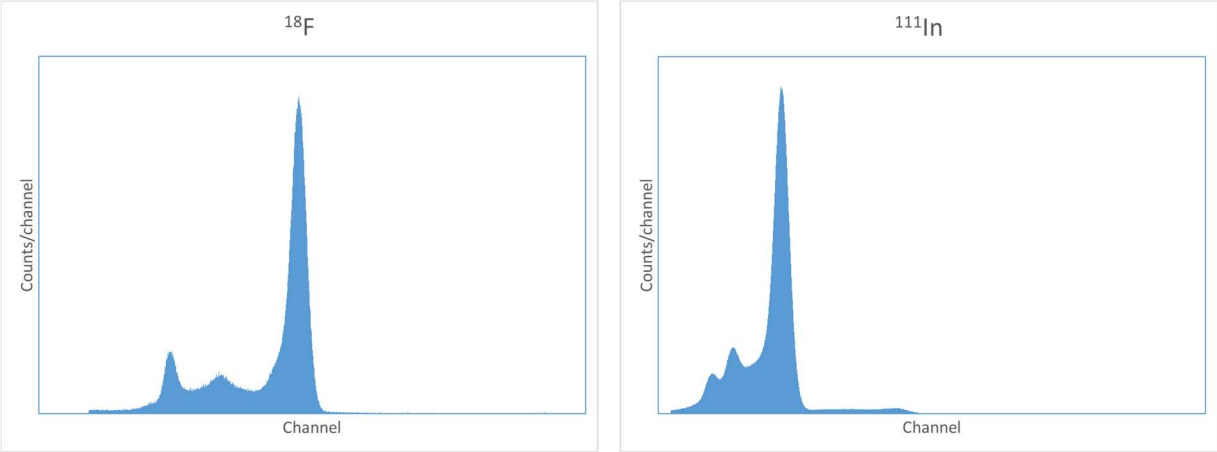
The demo code has two operating modes, set by the first parameter:

— **Encoding** (–e): Randomly sampled data is encoded into the IEC 63047 format and saved to a file.

— **Decoding** (–d): IEC 63047 formatted data is read from a file and printed on the screen.

In encoding mode, the code simulates the events recorded by a digital data acquisition instrument acquiring data from a NaI well-type detector. Every event contains a timestamp and a pulse height. The time between two successive events is randomly generated and determined by a parameter that represents the average count rate. The pulse height is randomly drawn from a histogram with 4096 channels. The demo code includes two histogram files of the radionuclides ^{111}In and ^{18}F, recorded on an 8" × 8" NaI well detector (Figure 2).

Figure 2. Example histograms that can be used by the demo code to generate simulated detector events.



9.2 Running the demo code in encoding mode

To run the code in encoding mode, type:

```
[…]$ ./lmdtest.x –e rate time n hist.txt out.coer
```

Where the parameters after `–e` have the following meaning:

— `rate`: the event rate, expressed in events per second as integer;

— `time`: the duration of the data acquisition, expressed in seconds real time as integer;

— `n`: the (maximum) number of events grouped in values of the `EventList` type;

— `hist.txt`: the histogram file to sample from;

— `out.coer`: the name of the output file for the IEC 63047 encoded data.

For example, the following command will generate an IEC 63047 binary data file with name In111.coer, with events sampled from histogram file In111.txt. The average event rate is 150 per second. The acquisition time is 10 seconds and a maximum 200 events are stored in every value of the `EventList` type:

```
[…]$ ./lmdtest.x –e 150 10 200 In111.txt In111.coer
```

In total, about 1500 events are expected. The file In111.coer will include 10 encodings of the `Listmodedata` type: one `Header` value, eight `EventList` values (seven containing 200 events and one with about 100 events) and one `Footer` value.

9.3 Running the demo code in decoding mode

To run the code in decoding mode, type:

```
[…]$ ./lmdtest.x –d in.coer
```

The parameter `in.coer` is the binary data file with IEC 63047 data. The result of the decoding process will be printed on the screen.

9.4 Modifying the demo code

The demo code may be used as a basis for your own IEC 63047 codec.

Required data elements are defined by value, optional data elements by reference (pointer). The pointer is NULL when the element is not present.

See the source code files `lmdtest.c`, `Example1.h` and `Example1.c` for instructions on how to use and change the demo code. Refer to http://lionet.info/asn1c/blog/ for information about `asn1c`.

10 Known issue with asn1c

10.1 Introduction

The version of asn1c shown in Table 1 does not provide correct decoding algorithms under certain conditions. The issue affects only the *decoding* (not the encoding) of the messageList element in the Header and of the syncStatus element in the Channel.

10.2 Avoiding the issue

Currently, the only solution is to avoid the issue by not using the messageList in the Header, and not using the syncStatus element in the Channel.

10.3 Detailed description of the issue

Refer to the ASN.1 standards in ([1]) and ([3]) for an explanation on the terminology.

In ASN.1 C-OER encoding, the presence of extensions and optional elements (or elements defined with a default value but assigned a non-default value) is indicated with an *extension bit* and a *preamble*. Whenever the ASN.1 specification contains an *extension mark* ('...'), and if at the same time the *preamble* occupies two octets, then the presence of the element indicated by the first bit of the second preamble octet is not correctly *decoded* by asn1c. The issue does not exist if there are two preamble octets without an extension bit.

The Header definition contains an extension mark and has eight optional/default elements (listModeDataID, listModeDataPart, listModeDataNParts, measSetupID, measSetupDescription, radSource, startAccuracy and messageList). The presence of an extension and each of the optional/default elements is indicated by nine bits, for which two preamble octets are required. The first bit of the second preamble octet is the bit that indicates the presence of the messageList. When the messageList is used in the header, the asn1c encoding of the header will be correct, but the asn1c decoder will fail.

Likewise, the syncStatus element in the (extensible) Channel definition corresponds to the first bit of the second preamble octet. When the syncStatus element is used, asn1c will correctly encode the data, but will be unable to decode.

11 Conclusions

IEC 63047 is a standard specifying the format of list-mode data acquired by nuclear data acquisition instruments. It is a binary format, defined using ASN.1. Several commercial solutions are available to automatically create source code for the encoding and decoding of data formats specified using ASN.1. This report provides an open-source solution to a limited number of IEC 63047 data types. The solution builds on the open-source code `asn1c` from GitHub, and includes demo source code developed by the JRC, which serves as the basis for the development of more elaborate code for the user's needs.

References

(1) IEC 63047, Edition 1.0, 2018-10, *Nuclear instrumentation – Data format for list mode digital data acquisition used in radiation detection and measurement*, International Electrotechnical Commission, Geneva, Switzerland

(2) ISO/IEC 8824-1, ITU-T X.680, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*, International Organization for Standardization, International Electrotechnical Commission and International Telecommunication Union, Geneva, Switzerland

(3) ISO/IEC 8825-7, ITU-T X.696, *Information technology – ASN.1 encoding rules: Specification of Octet Encoding Rules (OER)*, International Organization for Standardization, International Electrotechnical Commission and International Telecommunication Union, Geneva, Switzerland

(4) Lev Walking and contributors, *asn1c, an ASN.1 to C compiler*, https://github.com/vlm/asn1c .

(5) EUPL v.1.2, *Commission Implementing Decision (EU) 2017/863 of 18 May 2017 updating the open source software licence EUPL to further facilitate the sharing and reuse of software developed by public administrations* (http://data.europa.eu/eli/dec_impl/2017/863/oj)

(6) Commission Decision C(2021) 148 of 7 January 2021 on the distribution of IEC 63047 Codec software, Ref. Ares(2021)146054

List of abbreviations and definitions

ASN.1	Abstract Syntax Notation One
CBRNE	Chemical, Biological, Radiological, Nuclear and Explosives
C-OER	Canonical Octet Encoding Rule
DG HOME	European Commission's Directorate General for Migration and Home Affairs
EUPL	European Union Public Licence
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JRC	Joint Research Centre
NaI	Sodium-iodide

List of boxes

List of figures

List of tables

Annexes

Box 4. Note about source code

The source code in the annexes of this manual is also included as an attachment to the pdf file of this document.

Annex 1. Script `gen.sh`

```
#!/bin/bash
#
########################################
# Script to make official IEC63047.asn #
# compatible with asn1c                #
# and 'compile' with asn1c             #
# v20190315                            #
########################################
#
#
file=IEC63047.asn
fout=IEC63047LMT.asn
#
ftmp=.tmp.asn
#
if [ -e $ftmp ]; then
    \rm $ftmp
fi
#
echo -n "Searching for" $file "file"
if [ -e $file ]; then
    echo " .. found!"
else
    echo " ..  not found in" `pwd`
    exit
fi
#
if [ -e $fout ]; then
   echo
    echo $fout "already exists !!"
    echo "Please rename or delete it and restart this script!"
    exit
fi
#
lineid=`grep -n "date-time" $file | cut -d ":" -f 1`
line=`grep -n "date-time" $file | cut -d ":" -f 2`
if [ -z $lineid ]; then
    echo "date-time definition not found in" $file "!!"
    exit
fi
sed -s "${lineid} c\ date-time       LMTime," $file > $ftmp
#
lineid=$((lineid - 1))
#sed "${lineid}i\LMTime ::= SEQUENCE {\n \\-\\-TO MIMIC TIME in asn1c\n id INTEGER (0..2), \\-
\\-id must be set to 2, cannot set DEFAULT VALUE here otherwise omitted\n year INTEGER,\n
month INTEGER (1..12),\n day INTEGER (1..31),\n hour INTEGER (0..24),\n min INTEGER (0..59),\n
sec INTEGER (0..60)\n}\n" $ftmp > $fout
sed "${lineid}i\LMTime ::= SEQUENCE {\n year INTEGER,\n month INTEGER (1..12),\n day INTEGER
(1..31),\n hour INTEGER (0..24),\n min INTEGER (0..59),\n sec INTEGER (0..60)\n}\n" $ftmp >
$fout
#
\rm $ftmp
#
#
#
echo -n "Patched file:" $fout
echo "   Now file will be compiled with asn1c"
sleep 2
#
#
unameOut="$(uname -s)"
case "${unameOut}" in
    Linux*)     EXEC=asn1c;;
    CYGWIN*)    EXEC=asn1c.exe
esac
#
```

```
#
${EXEC} -gen-OER -fcompound-names ${fout}
sed -i 's/asn DEF Member 7/asn DEF NativeInteger/g' Footer.c
sed -i 's/asn DEF Member 9/asn DEF NativeInteger/g' Footer.c
#
#
if [ -e Makefile.am.asn1convert ]; then
    \rm Makefile.am.asn1convert
fi
if [ -e Makefile.am.libasncodec ]; then
    \rm Makefile.am.libasncodec
fi
if [ -e converter-example.c ]; then
    \rm converter-example.c
fi
if [ -e converter-example.mk ]; then
    \rm converter-example.mk
fi
#
#
echo
echo "moved generated files in folders:"
echo "     *.h in inc"
echo "     *.c in src, except lmdtest.c"
#
if [ -d inc ];then
    \mv -f *.h inc
else
    mkdir inc
    \mv -f *.h inc
fi
#
if [ -d src ];then
    \mv -f *.c src
else
    mkdir src
    \mv -f *.c src
fi
#
if [ -e "src/lmdtest.c" ];then
    \mv -f src/lmdtest.c .
fi
#
mkdir tmp
mkdir bin

echo " done"
```

Annex 2. JRC demo source code – file `lmdtest.c`

```
// This file is part of the IEC 63047 codec demo, developed by the European Commission's Joint
Research Centre
// The IEC 63047 codec demo is available under the conditions of the EUPL 1.2 licence
https://eupl.eu

#include <stdio.h>
#include <sys/types.h>
#include <sysexits.h>   /* for EX_* exit codes */
#include <math.h>
#include <Listmodedata.h>
#include <Example1.h>

// To enable debugging, add the following to asn_internal.h
// #define EMIT_ASN_DEBUG 1 // Enable debugging

// forward declarations
int WriteListmodedataToFile(const char *outfile, double *CHist, uint32 t rate, uint32 t
totalrealtime, uint32 t maxevents);
int AppendHeaderToFile(FILE *ofd, uint32_t tsclkfreq);
int AppendEventListToFile(FILE *ofd, uint32_t  id, uint32_t nEvents, uint32_t  *realtime,
uint32 t rate, double* CHist);
int AppendFooterToFile(FILE *ofd, long unsigned int LastEventListid);
int ReadListmodedataFromFile(const char *infile);
int InterpretListmodedata(Listmodedata_t* lmd);
int InterpretHeader(Header_t* h);
int InterpretEventList(EventList_t* e);
int InterpretFooter(Footer t* f);

/* Dump the buffer out to the specified FILE */
static int write_out(const void *buffer, size_t size, void *key) {
   FILE *fp = (FILE *)key;
    if (fp == stdout) {
        int i;
        for (i = 0; i < size; i++)
            fprintf(stdout, "%02X ", *((const unsigned char *)buffer + i));
        fprintf(stdout, "\n");
    } else {
       return (fwrite(buffer, 1, size, fp) == size) ? 0 : -1;
    }
}


int ReadSpectrum(const char* filename, double CHist[])
{
   // Read spectrum and return normalised cumulative histogram
   FILE *ptr file;

   uint32 t col0;
   uint32 t sum = 0;
   int row = 0;
   // uint32 t Hist[4096];
   uint32 t *Hist = (uint32 t*)malloc(4096*sizeof(uint32 t));

   ptr file = fopen(filename,"r");
   if (!ptr file)
      return 1;

   while ((row<4096) && (!feof(ptr file))){
      fscanf(ptr_file, "%lu",&Hist[row]);
      sum += Hist[row];
      row++;
   };

   CHist[0] = (double)Hist[0]/sum;
   for (row = 1; row < 4096; row++){
      CHist[row] = CHist[row-1] + (double)Hist[row]/sum;
   };

   free(Hist);
   fclose(ptr_file);

   return 0;
}

int main(int argc, char **argv) {
```

```
   int r;

   printf("\nIEC 63047 codec demo\n");
   printf("European Commission, Joint Research Centre\n");
   printf("Version 1.0\n");
   printf("Jan Paepen (jan.paepen@ec.europa.eu)\n\n");

   if ((argc <= 1) || (argc > 7)){
      printf ("To encode:\n");
      printf ("    ./lmdtest.x -e rate time n hist.txt out.coer\n");
      printf ("    rate: event rate (/s) - uint32, nonzero\n");
      printf ("    time: data acquisition duration, realtime (s) - uint32, nonzero\n");
      printf ("    n: number of events in each eventlist\n");
      printf ("    hist.txt: energy histogram to sample from\n");
      printf ("    our.coer: output file in the IEC 63047 format\n\n");

      printf ("To decode:\n");
      printf ("    ./lmdtest.x -d in.coer\n");
      printf ("    in.coer: input file in the IEC 63047 format\n");

      exit(EXIT FAILURE);
   }

   if (argv[1][1]=='e'){
      // ENCODE
      printf("ENCODE\n");
      uint32_t rate = 0;
      uint32 t totalrealtime = 0;
      uint32 t maxevents = 0;
      sscanf(argv[2],"%lu",&rate);
      sscanf(argv[3],"%lu",&totalrealtime);
      sscanf(argv[4],"%lu",&maxevents);

      // Remove these later. Only for debugging
      if (rate == 0) rate = 100;
      if (totalrealtime == 0) totalrealtime = 5;
      if (maxevents == 0) maxevents = 200;

      if ((rate == 0) || (totalrealtime == 0) || (maxevents == 0)) {
         printf("Invalid parameter value.\n");
         exit(EXIT FAILURE);
      }

      printf("rate: %lu /s\ntotal real time: %lu s\nmaxevents: %lu\nspectrum: %s\nout file:
%s\n",rate, totalrealtime, maxevents, argv[5], argv[6]);

      double *CHist = (double*)malloc(4096*sizeof(double));
      r = ReadSpectrum ((const char*)argv[5],CHist);

      // WRITE DATA INTO IEC 63047 FORMAT
      WriteListmodedataToFile((const char*)argv[6], CHist, rate, totalrealtime, maxevents);

      free(CHist);
   } else if (argv[1][1] == 'd') {
      // DECODE
      printf("DECODE\n");

      // READ IEC 63047 FORMATTED DATA
      ReadListmodedataFromFile((const char*)argv[2]);
   } else {
      printf("Invalid parameter value.\n");
      exit(EXIT_FAILURE);
   }

   exit(EXIT SUCCESS);
}

// Write listmode data to file
int WriteListmodedataToFile(const char *outfile, double *CHist, uint32 t rate, uint32 t
totalrealtime, uint32 t maxevents)
{
   // Open output file with encoded data
   FILE *ofd = NULL;
   ofd = fopen(outfile, "w"); // start with empty file
   fclose (ofd);
   ofd = fopen(outfile, "a"); // append
```

```
   if (ofd == NULL) {
      fprintf(stderr, "failed to open output file\n");
      exit(EX DATAERR);
   }

   // Determin an appropriare time stamp clock frequency, depending on the rate
   uint32_t tsclkfreq = (uint32_t)pow(10,ceil(log10((double)rate))+2); // attention: this
function is also used elsewhere

   printf("rate = %lu tsclockfreq = %lu\n", rate, tsclkfreq );

   // Append the header to the file
   // There shall be exaclty on header

   if (AppendHeaderToFile(ofd, tsclkfreq)) {
      perror("AppendHeaderToFile() failed");
      exit(EXIT_FAILURE);
   }

   // Append event lists to the file
   // There shall be at least one eventlist

   uint32_t totalevents = totalrealtime * rate;
   uint32_t eventsadded = 0; // Number of events so far
   uint32_t elID = 0; // Eventlist ID
   uint32 t realtime = 0; // Realtime
   uint32 t eventstoadd = 0; // Number of events to add
   double lambda = (double)rate;

   while (eventsadded < totalevents){
      // Determine number of events to add in this eventlist
      if (totalevents - eventsadded >= maxevents) {
         // still more than maxevents to generate
         eventstoadd = maxevents;
      } else {
         // less than maxevents left to generate
         eventstoadd = totalevents - eventsadded;
      };
      //printf("ID = %lu: %lu\n", elID, eventstoadd );

      if (AppendEventListToFile(ofd, elID, eventstoadd, &realtime, lambda, CHist)) {
         perror("AppendEventListToFile() failed");
         exit(EXIT FAILURE);
      };
      eventsadded += eventstoadd;
      elID++;
   }

   // Append the footer to the file
   // There shall be exaclty on footer
   if (AppendFooterToFile(ofd, elID-1)) {
      perror("AppendFooterToFile() failed");
      exit(1);
   }

   // Close the listmodedata file
   fclose (ofd);

   exit(EXIT SUCCESS);
}


int AppendHeaderToFile(FILE *ofd, uint32_t tsclkfreq){

   // Type to encode
   Listmodedata t *lmd;

   // Allocate space for Listmodedata_t
   lmd = calloc(1, sizeof(Listmodedata_t)); // not malloc!
   if (!lmd) {
      perror("calloc() failed");
      exit(1);
   };

   // Generate the header
   if (GenerateHeader(lmd, tsclkfreq)) {
      perror("GenerateHeader() failed");
```

```
      exit(1);
   }

   // Validate the target structure
   char *errbuf;
   size_t *errlen;
   if (asn_check_constraints(&asn_DEF_Listmodedata, lmd, errbuf, errlen )){
      // Failed
      printf("Constraint violation on header: %s\n",errbuf);
      exit(EX UNAVAILABLE);
   }

   // Print the target structure
   printf ("Data to encode:\n");
   asn fprint(stdout, &asn DEF Listmodedata, lmd);

   // Encode and write the output to file ofd
   asn_enc_rval_t ec; // encoder return value
   ec = oer_encode(&asn_DEF_Listmodedata, lmd, write_out, ofd);
   if (ec.encoded == -1){
      // Failed
      printf("Could not encode %s\n",ec.failed type->name);
      exit(EX_UNAVAILABLE);
   } else {
      // Success
      printf("Number of bytes encoded en written to file: %d\n",ec.encoded);
   }

   // Free the target structure
   ASN STRUCT FREE(asn DEF Listmodedata, lmd);

   return 0;
}


int AppendEventListToFile(FILE *ofd, uint32 t  id, uint32 t nEvents,
   uint32 t  *realtime, uint32 t rate, double* CHist ){

   // Type to encode
   Listmodedata_t *lmd;

   // Allocate space for Listmodedata t
   lmd = calloc(1, sizeof(Listmodedata t)); // not malloc!
   if (!lmd) {
      perror("calloc() failed");
      exit(1);
   };

   // Generate the event list
   if (GenerateEventList(lmd, id, nEvents, realtime, rate, CHist)) {
      perror("GenerateEventList() failed");
      exit(1);
   }

   // Validate the target structure
   char *errbuf;
   size_t *errlen;
   if (asn check constraints(&asn DEF Listmodedata, lmd, errbuf, errlen )){
      // Failed
      printf("Constraint violation on eventlist: %s\n",errbuf);
      exit(EX UNAVAILABLE);
   }

   // Print the target structure
   printf ("Data to encode:\n");
   asn fprint(stdout, &asn DEF Listmodedata, lmd);

   // Encode and write the output to file ofd
   asn_enc_rval_t ec; // encoder return value
   ec = oer encode(&asn DEF Listmodedata, lmd, write out, ofd); // OER
   if (ec.encoded == -1){
      // Failed
      printf("Could not encode %s\n",ec.failed type->name); // strerror(errno));
      exit(EX_UNAVAILABLE);
   } else {
      // Success
      printf("Number of bytes encoded en written to file: %d\n",ec.encoded);
```

```
   }

   // Free the target structure
   ASN STRUCT FREE(asn DEF Listmodedata, lmd);
   return 0;
}


int AppendFooterToFile(FILE *ofd, long unsigned int LastEventListid){
   // Type to encode
   Listmodedata_t *lmd;

   // Allocate space for Listmodedata_t
   lmd = calloc(1, sizeof(Listmodedata t)); // not malloc!
   if (!lmd) {
      perror("calloc() failed");
      exit(1);
   };

   // Generate the footer
   if (GenerateFooter(lmd,LastEventListid)) {
      perror("GenerateFooter() failed");
      exit(1);
   }

   // Validate the target structure
   char *errbuf;
   size_t *errlen;
   if (asn check constraints(&asn DEF Listmodedata, lmd, errbuf, errlen )){
      // Failed
      printf("Constraint violation on footer: %s\n",errbuf);
      exit(EX UNAVAILABLE);
   }

   // Print the target structure
   printf ("Data to encode:\n");
   asn fprint(stdout, &asn DEF Listmodedata, lmd);

   // Encode and write the output to file ofd
   asn_enc_rval_t ec; // encoder return value
   ec = oer encode(&asn DEF Listmodedata, lmd, write out, ofd);
   if (ec.encoded == -1){
      // Failed
      printf("Could not encode %s\n",ec.failed type->name); // strerror(errno));
      exit(EX UNAVAILABLE);
   } else {
      // Success
      printf("Number of bytes encoded en written to file: %d\n",ec.encoded);
   }

   // Free the target structure
   ASN STRUCT FREE(asn DEF Listmodedata, lmd);

   return 0;
}

// Read listmode data from file
int ReadListmodedataFromFile(const char *infile)
{
   // Read listmode data from file
   // The file should contain more that one listmodedata value:
   // - One Header
   // - At least one EventList
   // - One footer
   // We assume to have enough memory to read the whole file

   // Open the file and read its contents
   printf("\nReading encoded data from file\n");
   FILE *ifd = fopen (infile,"rb"); // Open file for binary read
   if (ifd == NULL){
      fprintf(stderr, "failed to open input file\n");
      exit(EX DATAERR);
   }
   fseek(ifd, 0, SEEK_END); // find the end
   long fsize = ftell(ifd); // get the size
   fseek(ifd, 0, SEEK SET); // go back to first byte
```

```
   char *fullencodeddata = malloc(fsize+1); // get a chunk of memory, enough to store the
whole file (and a NULL at the end)
   if (!fullencodeddata) {
      perror("malloc() failed: not enough memory to read the complete file");
      exit(1);
   };

   char *encodeddata = fullencodeddata; // Keep the reference to the beginning
   fread(encodeddata, fsize, 1, ifd); // read the file
   fclose (ifd); // close the input file
   encodeddata[fsize] = 0; // add the NULL character at the end

   // Keep on decoding the buffer until it is empty
   long decodercalls = 0; // number of times decoder was called
   asn dec rval t rval;
   char *errbuf;
   size_t *errlen;
   while (fsize > 0)
   {
      printf("Filesize = %lu octets\n",fsize);
      Listmodedata t *decodedlmd = 0; // Decoded listmodedata. Note the 0!
      rval = oer decode(0,&asn DEF Listmodedata, (void**)&decodedlmd, encodeddata, fsize);

      if (rval.code == RC_OK) {
         // Decoding succeeded
         decodercalls++;
         printf("Decoding succeeded\n");

         // Validate the decoded data
         // Hangs in a loop when listModeDataID =  NULL in EventList or Footer

         if (asn check constraints(&asn DEF Listmodedata, decodedlmd, errbuf, errlen )){
            // Failed
            printf("Constraint violation: %s\n",errbuf);
            exit(EX UNAVAILABLE);
         } else {
            printf("Constraint checked and OK\n");
         }


         // Print the decoded data
         printf ("Decoded data (%lu bytes consumed):\n", rval.consumed); // rval.consumed
seems to report one byte too few
         asn fprint(stdout, &asn DEF Listmodedata, decodedlmd);

         // Read some elements
         InterpretListmodedata(decodedlmd);

         // Free the target structure
         ASN STRUCT FREE(asn DEF Listmodedata, decodedlmd);

         // Advance buffer
         fsize = fsize - rval.consumed ; // rval.consumed seems to report one byte too few
         encodeddata = &encodeddata[rval.consumed];

      } else {
         printf("Decoding failed\n");
         // Free patrially decoded data
         ASN STRUCT FREE(asn DEF Listmodedata, decodedlmd);
         return 0;
      }

   }

   // free data reserved with malloc
   free (fullencodeddata);

   printf("Number of listmodedata values found = %lu\n",decodercalls);
}


// Interpret Listmodedata
int InterpretListmodedata(Listmodedata t* lmd) {
   // What are we reading?
   switch(lmd->present) {
      case Listmodedata PR NOTHING:
         perror("CHOICE in Listmodedata not determined");
```

```
         exit(1);
         break;
      case Listmodedata_PR_header:
         return InterpretHeader(&lmd->choice.header);
      case Listmodedata_PR_eventList:
         return InterpretEventList(&lmd->choice.eventList);
      case Listmodedata_PR_footer:
         return InterpretFooter(&lmd->choice.footer);
      default:
         perror("Invalid CHOICE in Listmodedata");
         exit(1);
   }
   return 0;
}

// Interpret Header
int InterpretHeader(Header_t* h) {
   printf ("Header found\n");
}


// Interpret EventPulse
int InterpretEventPulse(EventPulse_t *pulse) {

   printf(".channelID: %lu\n",pulse->channelID);

   if (pulse->timeStamp == NULL) {
      printf(".timeStamp: (not present)\n");
   } else {
      printf(".timeStamp: %lu\n",pulse->timeStamp); // check %
   };

   if (pulse->valueList == NULL) {
      printf(".valueList: (not present)\n");
   } else
   {
      for (unsigned long i = 0; i < pulse->valueList->list.count; i++) {
         switch (pulse->valueList->list.array[i]->present) {

            case (Numeric_PR_NOTHING):
               perror("CHOICE in valueList not determined");
               exit(1);
               break;

            case (Numeric PR int):
               printf(".value %lu (int): %f\n",i,pulse->valueList->list.array[i]->choice.Int);
               break;

            case (Numeric PR real32):
               printf(".value %lu (real32): %f\n",i,pulse->valueList->list.array[i]-
>choice.real32);
               break;

            case (Numeric PR real64):
               printf(".value %lu (real64): %f\n",i,pulse->valueList->list.array[i]-
>choice.real64);
               break;

            default:
               perror("Invalid CHOICE in Eventlist");
               exit(1);
         }
      }
   }

   if (pulse->flags == NULL) {
      printf(".flags: (not present)\n");
   } else
   {

   }

}


// Interpret EventList
int InterpretEventList(EventList_t* e) {
```

```
   printf ("EventList found, with the following elements:\n");

   if (e->listModeDataID == NULL){
      printf(".listModeDataID: (not present)\n");
   } else {
      printf(".listModeDataID: ");
      UTF8String_print(&asn_DEF_UTF8String, e->listModeDataID, 0, write_out, stdout);
   };

   printf(".listModeDataPart: %lu\n",e->listModeDataPart);

   printf(".id: %lu\n",e->id);

   if (e->eventList.list.count == 0){
      perror("Eventlist is empty");
      exit(1);
   } else {
      for (unsigned long i = 0; i < e->eventList.list.count; i++){
         switch (e->eventList.list.array[i]->present) {

            case Event PR NOTHING:
               perror("CHOICE in Eventlist not determined");
               exit(1);
               break;

            case Event PR eventPulse:
               InterpretEventPulse(&e->eventList.list.array[i]->choice.eventPulse);
               break;

            case Event PR eventDigitalSignalList:
               // Add code here to interpret an event of the eventDigitialSignalList type
               break;

            case Event_PR_eventGeo:
               // Add code...
               break;

            case Event PR eventHistogram1DList:
               // Add code...
               break;

            case Event PR eventHistogram2DList:
               // Add code...
               break;

            case Event_PR_eventLogic:
               // Add code...
               break;

            case Event PR eventMeasurementList:
               // Add code...
               break;

            case Event PR eventMessage:
               // Add code...
               break;

            case Event PR eventRollover:
               // Add code...
               break;

            case Event_PR_eventRTC:
               // Add code...
               break;

            case Event PR eventTime:
               // Add code...
               break;

            default:
               perror("Invalid CHOICE in Eventlist");
               exit(1);
         }
      }
   };
}
```

```
// Interpret Footer
int InterpretFooter(Footer_t* f) {

   printf ("Footer found, with the following elements:\n");
   long i;

   if (f->listModeDataID == NULL){
      printf(".listModeDataID: (not present)\n");
   } else {
      printf(".listModeDataID: ");
      UTF8String_print(&asn_DEF_UTF8String, f->listModeDataID, 0, write_out, stdout);
   };

   printf(".listModeDataPart: %lu\n",f->listModeDataPart);

   printf(".lastEventListid: %lu\n",f->lastEventListid);

   if (f->stop == NULL){
      printf(".stop: (not present)\n");
   } else {
      printf(".stop.date time: %ld\n",f->stop->date time);
      printf(".stop.fractional seconds: %.9f\n", f->stop->fractional seconds); // TODO: does
not seem to be correct
   };

   if (f->totalDeadTimeList == NULL) {
      printf(".totalDeadTimeList: (not present)\n");
   } else {
      printf(".totalDeadTimeList:\n");
      for (i = 0; i < f->totalDeadTimeList->list.count; i++)
         printf(" %ld\n",f->totalDeadTimeList->list.array[i][0]);
   };

   if (f->totalLiveTimeList == NULL) {
      printf(".totalLiveTimeList: (not present)\n");
   } else {
      printf(".totalLiveTimeList:\n");
      for (i = 0; i < f->totalLiveTimeList->list.count; i++)
         printf(" %ld\n",f->totalLiveTimeList->list.array[i][0]);
   };


   return 0;
}
```

Annex 3. JRC demo source code – file `Example1.h`

```
// This file is part of the IEC 63047 codec demo, developed by the European Commission's Joint
Research Centre
// The IEC 63047 codec demo is available under the conditions of the EUPL 1.2 licence
https://eupl.eu

int GenerateHeader(Listmodedata t* lmd, uint32 t tsclkfreq);
int GenerateEventList(Listmodedata_t* lmd, uint32_t id, uint32_t nEvents, uint32_t *realtime,
uint32_t rate, double* CHist);
int GenerateFooter(Listmodedata_t* lmd, long unsigned int LastEventListid);
```

Annex 4. JRC demo source code – file `Example1.c`

```
// This file is part of the IEC 63047 codec demo, developed by the European Commission's Joint
Research Centre
// The IEC 63047 codec demo is available under the conditions of the EUPL 1.2 licence
https://eupl.eu
//
// IEC 63047 examples
//
// Example 1: one device, one channel, timestamp and pulse height, sampled from a spectrum
// ------------------------------------------------------------------------------------------
// It is recommended to verify the constraints in code, and not only rely on asn1c.


static const char *lmdID = "Example 1";

#include <math.h>
#include <time.h>
#include <Listmodedata.h>


uint32_t SampleSpectrum(double* CHist){
   // Sample the normalised cumulatice histogram
   double rnd = (double)rand()/RAND MAX; // random number between 0 and 1
   int c = 0; // channel
   while ((CHist[c] <= rnd) && (c < 4096)) {
      c++;
   };
   return c;
}


// Gnerate the header
int GenerateHeader(Listmodedata_t* lmd, uint32_t tsclkfreq) {
   int rv; // return value of some functions

   lmd->present = Listmodedata PR header;

   ////////////////////////////////
   // Header element "standardID" //
   ////////////////////////////////
   // Note:this is a fixed-value string that says that this file is compliant with IEC 63047
Ed. 1.0
   // Note that this is not a default value. It shall be provided before encoding.
   // Note that VisibleString is implemted via OCTET_STRING
   VisibleString t *stdID = OCTET STRING new fromBuf(&asn DEF VisibleString, "IEC 63047 Ed.
1.0", -1);
   lmd->choice.header.standardID =  *stdID;

   ////////////////////////////////////
   // Header element "listModeDataID" //
   ////////////////////////////////////
   // Defined by reference, OPTIONAL, NULL when not used
   // Identifies the list-mode data
   // Note that UTF8String is implemted via OCTET_STRING
   lmd->choice.header.listModeDataID = OCTET STRING new fromBuf(&asn DEF UTF8String, lmdID, -
1);

   ////////////////////////////////////
   // Header element "listModeDataPart" //
   ////////////////////////////////////
   // Defined by value
   // List-mode data part number.
   // Used when list-mode data is split in more than one file or stream.
   // When used, also listModeDataNParts shall be used.
   // The default is 0. When set to 0, it will not be encoded.
   // Will be decoded as 0 when not set.
   // Hence, only values not 0 will be encoded
   lmd->choice.header.listModeDataPart = 0;

   ////////////////////////////////////////
   // Header element "listModeDataNParts" //
   ////////////////////////////////////////
   // Number of parts in the list-mode data
   // Default = 1
   // Used when list-mode data is split in more than one file or stream.
   // When used, also listModeDataPart shall be used.
```

```
   long *NParts = calloc(1,sizeof(long));
   *NParts = 1;
   lmd->choice.header.listModeDataNParts = NParts;

   ////////////////////////////////
   // Header element "measSetupID" //
   ////////////////////////////////
   // Defined by reference, OPTIONAL, NULL when not used
   // Identifies the measurement setup
   // Note that UTF8String is implemted via OCTET STRING
   lmd->choice.header.measSetupID = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "NaI well
6", -1);

   ////////////////////////////////////////
   // Header element "measSetupDescription" //
   ////////////////////////////////////////
   // Defined by reference, OPTIONAL, NULL when not used
   // String that contains a description of the measurement setup.
   // Note that UTF8String is implemted via OCTET_STRING
   lmd->choice.header.measSetupDescription = OCTET STRING new fromBuf(&asn DEF UTF8String,
"Description goes here", -1);

   ////////////////////////////////
   // Header element "iec62755" //
   ////////////////////////////////
   // Defined by value
   // Use to specify the relation with an IEC 62755 file.
   // One of the four possible options are valid:

   // IEC62755 PR single : choice.single is NULL
   lmd->choice.header.iec62755.present = IEC62755 PR single;
   lmd->choice.header.iec62755.choice.single = 0;

   // IEC62755_PR_coexisting : choice.coexisting is of the UTF8String_t type
   // lmd->choice.header.iec62755.present = IEC62755 PR coexisting;
   // UTF8String t *coexist = OCTET STRING new fromBuf(&asn DEF UTF8String, "Co-existing with
xxx", -1);
   // lmd->choice.header.iec62755.choice.coexisting = *coexist;

   // IEC62755_PR_embedded : choice.embedded is of the UTF8String_t type
   // lmd->choice.header.iec62755.present = IEC62755 PR embedded;
   // UTF8String t *embed = OCTET STRING new fromBuf(&asn DEF UTF8String, "Embedded in xxx", -
1);
   // lmd->choice.header.iec62755.choice.embedded = *embed;

   // IEC62755_PR_included : choice.included is of the OCTEC_STRING_t type
   // lmd->choice.header.iec62755.present = IEC62755 PR included;
   // OCTET STRING t *included = OCTET STRING new fromBuf(&asn DEF UTF8String, "Included in
xxx", -1);
   // lmd->choice.header.iec62755.choice.included = *included;

   ////////////////////////////////
   // Header element "radSource" //
   ////////////////////////////////
   // Defined by reference, OPTIONAL, NULL when not used
   // Identifies the radiation source.
   // Note that UTF8String is implemented via OCTET_STRING
   lmd->choice.header.radSource = OCTET STRING new fromBuf(&asn DEF UTF8String, "Sampled from
a spectrum", -1);

   ////////////////////////////
   // Header element "start" //
   ////////////////////////////
   // Defined by value
   // Defines the reference for all timestamps.
   // Is the start of the acquisition process.
   // Shall be specified in UTC.
   LMTime_t startdt;
   startdt.year = 2018;
   startdt.month = 10;
   startdt.day = 19;
   startdt.hour = 14;
   startdt.min = 12;
   startdt.sec = 10;
   lmd->choice.header.start.date_time = startdt;
   lmd->choice.header.start.fractional seconds = .035;
```

```
   ////////////////////////////////////
   // Header element "startAccuracy" //
   ////////////////////////////////////
   // Accuracy of start expressed in seconds
   // Defined by reference, NULL when not used
   REAL64_t *startAcc = calloc(1,sizeof(REAL64_t));
   *startAcc = .12345678;
   lmd->choice.header.startAccuracy = startAcc;

   ///////////////////////////////////
   // Header element "deviceList" //
   ///////////////////////////////////
   // Defined by value
   // List of devices that take part in the data acquisition process.
   // Position of device in the sequence defines the deviceID.
   // deviceID = 0 for the first device in the list.
   // Increments with 1 for every additional device.
   // There can be a maximum of 256 devices in one list-mode data file or stream.
   // Over all list-mode data parts, there can be 1 to 65536 devices.
   // A device has 4 optional elements: name, manuf, model and serial.
   // Optional elements are defined via pointers. NULL means that no value is assigned to the
element.

   // Declare pointer to a device
   Device_t *deviceid0 = calloc(1,sizeof(Device_t));
   if (!deviceid0) {
      perror("calloc() failed");
      exit(1);
   };

   // Set the values
   deviceid0->name = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "DigitiserID=01", -1);
   deviceid0->manuf = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "ManufacturerDEF", -1);
   deviceid0->model = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "ModelGHI", -1);
   deviceid0->serial = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "S/N JKL123", -1);

   // Add the device to the list
   if (ASN_SEQUENCE_ADD(&lmd->choice.header.deviceList,deviceid0)) {
      perror("ASN_SEQUENCE_ADD() failed device");
      exit(1);
   };

   ///////////////////////////////////
   // Header element "channelList" //
   ///////////////////////////////////
   // Defined by value
   // Lists all channels that take part in the data acquisition process.
   // Position of channel in the sequence defines the channelID.
   // channelID = 0 for the first channel in the list.
   // Increments with 1 for every additional channel.
   // There is 1 to 265 channels in each device.

   // Generate a channel (with id = 0)
   Channel_t *channelid0 = calloc(1,sizeof(Channel_t));
   if (!channelid0) {
      perror("calloc() failed");
      exit(1);
   };

   // Set the values of the channel

   // Identifies the Device.
   // Default = 0. Defined in Channel.h by value
   channelid0->deviceID = 0;

   // Channel_kind_virtualkind or Channel_kind_physicalkind
   // Defined in Channel.h by value
   channelid0->kind = Channel__kind_physicalkind;

   // Identifies the physical channel on the device, omit when virtual
   // Defined in Channel.h by reference
   // Value shall not occupy more than 8 octets
   // Is an OPTIONAL element: NULL when not provided, pointing to a value when provided
   channelid0->physicalChannel = calloc(1,sizeof(unsigned long));
   *channelid0->physicalChannel = (unsigned long)6;

   // Name of the channel
```

```
   // Defined in Channel.h by reference
   // OPTIONAL: NULL when not used
   UTF8String_t *channelid0name = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "Channel 0", -
1);
   channelid0->name = channelid0name;

   // Description of the channel
   // Defined in Channel.h by reference
   // OPTIONAL: NULL when not used
   UTF8String_t *channelid0descr = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "A
description of channel 0", -1);
   channelid0->description = channelid0descr;

   // Parameters used to set up or configure the channel
   // Defined in Channel.h by reference
   // OPTIONAL: NULL when not used
   UTF8String_t *channelid0params = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "Parameters
for channel 0", -1);
   channelid0->parameters = channelid0params;

   // Delay applied to all timestamps of the channel
   // Defined in Channel.h by value
   // Expressed as an integer multiple of time base
   // Value shall not occupy more than 8 octets
   // Default = 0
   channelid0->delay = -547;

   // Reference clock to which the channel is synchronised
   // Defined in Channel.h by reference
   // OPTIONAL: NULL when not used
   UTF8String_t *channelid0refClock = OCTET_STRING_new_fromBuf(&asn_DEF_UTF8String, "DCF-77
receiver in lab 13", -1);
   channelid0->refClock = channelid0refClock;

   // Status of synchronisation with reference clock at start
   // SyncStatus synchronised or SyncStatus unsynchronised
   // Defined in Channel.h by reference
   // OPTIONAL: NULL when not used
   //SyncStatus_t *syncst = calloc(1,sizeof(SyncStatus_t));
   //*syncst  = SyncStatus_synchronised;
   // asn1c is able to encode, but fails to decode the element when it is used
   // In ASN.1 C-OER encoding, the presence of extensions and optional elements
   // (or elements defined with a default value but assigned a non-default value)
   // is indicated with an extension bit and a preamble. Whenever the ASN.1 specification
   // contains an extension mark ("..."), and if at the same time the preamble occupies
   // two octets, then the presence of the element indicated by the first bit of the
   // second preamble octet is not correctly decoded by asn1c.
   // The issue does not exist if there are two preamble octets without an extension bit.

   channelid0->syncStatus = NULL;

   // Sample rate of the ADC, in samples per second
   // Defined in Channel.h by value
   // Positive
   // Should be INTEGER if possible
   // Integer 0 for conventional ADCs
   Numeric_t channelid0adcSamplingRate;
   channelid0adcSamplingRate.present = Numeric_PR_int;
   channelid0adcSamplingRate.choice.Int = 100000000;
   channelid0->adcSamplingRate = channelid0adcSamplingRate;

   // Number of bits of the ADC
   // Defined in Channel.h by value
   channelid0->adcBitResolution = 12; // 12 bits ADC

   // Jitter of the ADC sampling process, expressed in RMS seconds
   // Defined in Channel.h by reference
   // NULL when not used
   REAL32_t *adcjit = calloc(1,sizeof(REAL32_t));
   *adcjit = 0.6e-9;
   channelid0->adcJitterRMS = adcjit;

   // Jitter on the timestamp, expressed in RMS seconds
   // Defined in Channel.h by reference
   // NULL when not used
   REAL32_t *tsjit = calloc(1,sizeof(REAL32_t));
   *tsjit = 0.07e-9;
```

```
   channelid0->timeStampJitterRMS = tsjit;

   // Number of bits consumed by FINE TIME, when specified as INTEGER
   // Defined in Channel.h by reference
   // NULL when not used
   channelid0->fineTimeBitResolution = calloc(1,sizeof(long));
   *channelid0->fineTimeBitResolution = 10;

   // clockFrequency: Number of timestamp counter increments per second (inverse of time base)
   // Defined in Channel.h value
   // Positive
   // Should be INTEGER if possible
   Numeric_t channelid0clockFrequency;
   channelid0clockFrequency.present = Numeric PR int;
   channelid0clockFrequency.choice.Int = tsclkfreq;
   channelid0->clockFrequency = channelid0clockFrequency;


   ////////////////////////////////////////
   // Channel element "eventPropertyList" //
   ////////////////////////////////////////
   // Defined in Channel.h by value
   // Contains the properties of all the events written by the channel

   // Generate an event property
   EventProperty t *channelid0eventProperty0 = calloc(1,sizeof(EventProperty t));
   if (!channelid0eventProperty0) {
      perror("calloc() failed");
      exit(1);
   };

   // Set the CHOICE of event property
   // This channel can produce Events of the EventPulse type.
   channelid0eventProperty0->present = EventProperty_PR_eventPulseProperty;

   // Generate an event pulse property
   EventPulseProperty t *pulseProp0 = calloc(1,sizeof(EventPulseProperty t));
   if (!pulseProp0) {
      perror("calloc() failed");
      exit(1);
   };

   // description
   // OPTIONAL: NULL when not used
   pulseProp0->description = OCTET STRING new fromBuf(&asn DEF UTF8String, "Channel 0 records
TimeStamp and pulse height", -1); ;

   // valueDescriptionList
   // SEQUENCE OF UTF8String OPTIONAL
   UTF8String t *pulseProp0valueDescr0 = OCTET STRING new fromBuf(&asn DEF UTF8String, "Pulse
height", -1);

   // The valueDescriptionList is optional. Before adding items to the list, create an empty
list
   pulseProp0->valueDescriptionList = calloc(1,sizeof(struct
EventPulseProperty__valueDescriptionList));
   if (!pulseProp0->valueDescriptionList) {
      perror("calloc() failed");
      exit(1);
   };

   if (ASN_SEQUENCE_ADD(pulseProp0->valueDescriptionList, pulseProp0valueDescr0)) {
      perror("ASN_SEQUENCE_ADD() failed");
      exit(1);
   };

   // Connect the pulse property to the channels eventproperty
   channelid0eventProperty0->choice.eventPulseProperty = *pulseProp0;

   // Add the event property to the list of events that channel 0 can generate
   if (ASN SEQUENCE ADD(&channelid0->eventPropertyList,channelid0eventProperty0)) { // WORKS.
COUNT = 1, SIZE = 4
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };

      // Add the channel with ID = 0 to the ChannelList in the header
```

```
   if (ASN_SEQUENCE_ADD(&lmd->choice.header.channelList,channelid0)) {
      perror("ASN_SEQUENCE_ADD() failed");
      exit(1);
   };

   ////////////////////////////////
   // Header element "messageList" //
   ////////////////////////////////
   // List of text or binary messages or files to include in the header
   // Defined by reference
   // asn1c is able to encode, but fails to decode
   // In ASN.1 C-OER encoding, the presence of extensions and optional elements
   // (or elements defined with a default value but assigned a non-default value)
   // is indicated with an extension bit and a preamble. Whenever the ASN.1 specification
   // contains an extension mark ("..."), and if at the same time the preamble occupies
   // two octets, then the presence of the element indicated by the first bit of the
   // second preamble octet is not correctly decoded by asn1c.
   // The issue does not exist if there are two preamble octets without an extension bit.

   /*
   // Create a message
   EventMessage t *msg = calloc(1,sizeof(EventMessage t));
   if (!msg) {
      perror("calloc() failed");
      exit(1);
   };

   msg->channelID = calloc(1,sizeof(unsigned long));
   *msg->channelID = (unsigned long)0;
   msg->txtMessage = OCTET STRING new fromBuf(&asn DEF UTF8String, "This is a message:
Acquisition started!", -1);

   // The messagelist is optional. When needed, first create an empty list
   lmd->choice.header.messageList = calloc(1,sizeof(struct Header__messageList));
   if (!lmd->choice.header.messageList) {
      perror("calloc() failed");
      exit(1);
   };

   // Add the message to the list
   if (ASN SEQUENCE ADD(lmd->choice.header.messageList,msg)) {
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };
   */

   return 0;
}

// Gnerate an eventlist of example 1
int GenerateEventList(Listmodedata_t* lmd, uint32_t id, uint32_t nEvents, uint32_t *realtime,
uint32 t rate, double* CHist) {

   lmd->present = Listmodedata PR eventList;

   ////////////////////////////////////////
   // EventList element "listModeDataID" //
   ////////////////////////////////////////
   // Identifies the list-mode data
   // Optional: NULL when not used
   lmd->choice.eventList.listModeDataID = OCTET STRING new fromBuf(&asn DEF UTF8String, lmdID,
-1);

   ////////////////////////////////////////
   // EventList element "listModeDataPart" //
   ////////////////////////////////////////
   // INTEGER (0..255) DEFAULT 0
   // List-mode data part number
   // Defined by value
   lmd->choice.eventList.listModeDataPart = 0;

   ////////////////////////////
   // EventList element "id" //
   ////////////////////////////
   // INTEGER (0..MAX)
   // Identifies the EventList (from 0, incremented by one)
   // Value shall not occupy more than 8 octets
```

```
   // Defined by value
   lmd->choice.eventList.id = id;

   ////////////////////////////////
   // EventList element "eventList" //
   ////////////////////////////////
   // SEQUENCE (SIZE(1..MAX)) OF Event
   // Contains a list of events

   /* NOT NEEDED
   // Generate space for an empty eventlist
   struct EventList__eventList *evlist = calloc(1, sizeof(struct EventList__eventList)); //
Allocate space
   if (!evlist) {
      perror("calloc() failed");
      exit(1);
   };
   lmd->choice.eventList.eventList = *evlist;
   */

   // timestamp clock frequency; warning also used elsewhere
   uint32 t tsclkfreq = (uint32 t)pow(10,ceil(log10((double)rate))+2);
   srand ( time ( NULL));

   long ev;
   for (ev = 0; ev < nEvents; ev++)
   {
      // Generate space for an event
      Event t *event = calloc(1,sizeof(Event t));
      if (!event) {
         perror("calloc() failed");
         exit(1);
      };
      // Add the event to the eventlist
      if (ASN SEQUENCE ADD(&lmd->choice.eventList.eventList,event)) {
         perror("ASN SEQUENCE ADD() failed");
         exit(1);
      }

      // Generate a pulse height and a timestamp
      double rnd = (double)rand()/RAND MAX; // random number between 0 and 1
      double dt s = -log(rnd)/(double)rate; // time until next decay, in seconds

      uint32 t deltat =  (uint32 t)(dt s * (double)tsclkfreq);

      *realtime += deltat;

      // Specify the type of the event
      event->present = Event PR eventPulse; // This channel generates events of the EventPulse
type
      // Allocate space for the pulse
      EventPulse t *pulse = calloc(1,sizeof(EventPulse t));
      if (!pulse) {
         perror("calloc() failed");
         exit(1);
      };

      // Specify the channel ID
      pulse->channelID = 0;

      // Specify the timestamp
      pulse->timeStamp = calloc(1,sizeof(TimeStamp_t));
      *pulse->timeStamp =  (TimeStamp_t)*realtime;

      // specify the pulse height
      // Make space for the list of values
      struct EventPulse  valueList *valuelist = calloc (1,sizeof(struct
EventPulse__valueList));
      if (!valuelist) {
         perror("calloc() failed");
         exit(1);
      };
      pulse->valueList = valuelist; // Connect list to pulse
      // Make space for the value
      struct Numeric *value0 = calloc (1,sizeof(struct Numeric));
      if (!value0) {
         perror("calloc() failed");
```

```
         exit(1);
      };
      value0->present = Numeric PR int;

      value0->choice.Int = SampleSpectrum(CHist); //pulseheight;


      // Add the value to the list of values
      if (ASN SEQUENCE ADD(pulse->valueList,value0)) {
         perror("ASN SEQUENCE ADD() failed");
         exit(1);
      }

      // Connect the pulse to the event
      event->choice.eventPulse = *pulse;
   }

   return 0;
}


// Generate the footer of example 1
int GenerateFooter(Listmodedata_t* lmd, long unsigned int LastEventListid) {

   lmd->present = Listmodedata PR footer;

   /////////////////////////////////////
   // Footer element "listModeDataID" //
   /////////////////////////////////////
   // UTF8String OPTIONAL
   // Note that UTF8String is implemted via OCTET STRING
   lmd->choice.footer.listModeDataID = OCTET STRING new fromBuf(&asn DEF UTF8String, lmdID, -
1);

   ////////////////////////////////////////
   // Footer element "listModeDataPart" //
   ////////////////////////////////////////
   // INTEGER (0..255) DEFAULT 0
   lmd->choice.footer.listModeDataPart = 0;

   /////////////////////////////////////
   // Footer element "lastEventListid" //
   /////////////////////////////////////
   // INTEGER (0..MAX)
   // Required element
   lmd->choice.footer.lastEventListid = LastEventListid;

   ////////////////////////////////////////
   // Footer element "totalDeadTimeList" //
   ////////////////////////////////////////
   //
   // TO REVISE
   //
   // totalDeadTimeList SEQUENCE OF INTEGER (0..MAX) OPTIONAL
   //
   /*
   lmd->choice.footer.totalDeadTimeList = calloc(1, sizeof(A_SEQUENCE_OF(long))); // Allocate
space
   if (!lmd->choice.footer.totalDeadTimeList) {
      perror("calloc() failed");
      exit(1);
   };
   // Add three values
   long *totaldeadtime1 = calloc(1,sizeof(long));
   *totaldeadtime1 = 901;
   if (ASN SEQUENCE ADD(lmd->choice.footer.totalDeadTimeList,totaldeadtime1)) {
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };
   long *totaldeadtime2 = calloc(1,sizeof(long));
   *totaldeadtime2 = 902;
   if (ASN SEQUENCE ADD(lmd->choice.footer.totalDeadTimeList,totaldeadtime2)) {
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };
   long *totaldeadtime3 = calloc(1,sizeof(long));
   *totaldeadtime3 = 903;
```

```
   if (ASN_SEQUENCE_ADD(lmd->choice.footer.totalDeadTimeList,totaldeadtime3)) {
      perror("ASN_SEQUENCE_ADD() failed");
      exit(1);
   };

   */
   ////////////////////////////////////
   // Footer element "totalLiveTimeList" //
   ////////////////////////////////////
   //
   // TO REVISE
   //
   // totalLiveTimeList SEQUENCE OF INTEGER (0..MAX) OPTIONAL
   //
   /*
   lmd->choice.footer.totalLiveTimeList = calloc(1, sizeof(A SEQUENCE OF(long))); // Allocate
space
   if (!lmd->choice.footer.totalLiveTimeList) {
      perror("calloc() failed");
      exit(1);
   };
   // Add three values
   long *totallivetime1 = calloc(1,sizeof(long));
   *totallivetime1 = 99901;
   if (ASN SEQUENCE ADD(lmd->choice.footer.totalLiveTimeList,totallivetime1)) {
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };
   long *totallivetime2 = calloc(1,sizeof(long));
   *totallivetime2 = 99902;
   if (ASN SEQUENCE ADD(lmd->choice.footer.totalLiveTimeList,totallivetime2)) {
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };
   long *totallivetime3 = calloc(1,sizeof(long));
   *totallivetime3 = 99903;
      if (ASN SEQUENCE ADD(lmd->choice.footer.totalLiveTimeList,totallivetime3)) {
      perror("ASN SEQUENCE ADD() failed");
      exit(1);
   };
   */
   return 0;
}
```

Annex 5. The `Makefile` to build the executable of the JRC demo code

```
CC1=gcc
CFLAGS1=-std=gnu99 -O2 -Wall -march=native
MAIN1:=lmdtest.c
EXE1:=$(MAIN1:.c=.x)
BINEXE1:=bin/$(EXE1)
#
#CC2=gcc
#CFLAGS2=-std=gnu99 -O2 -Wall -march=native
#MAIN2:=coer2ascii.c
#EXE2:=$(MAIN2:.c=.x)
#BINEXE2:=bin/$(EXE2)
#
#CC3=g++
#MAIN3:=coer2root.c
#EXE3:=$(MAIN3:.c=.x)
#BINEXE3:=bin/$(EXE3)
#CFLAGS3=-O2 -Wall -march=native

#
#MAIN4:=coer2root_stream.c
#EXE4:=$(MAIN4:.c=.x)
#BINEXE4:=bin/$(EXE4)
#
#CFLAGS= -std=gnu99 -O2 -Wall #-mcpu=cortex-a53 -mtune=cortex-a53 -mfpu=neon-fp-armv8 -mfloat-
abi=hard -funroll-loops #-Q
LIBS= -lm
#
SRC := $(filter-out src/converter-example.c, $(wildcard src/*.c))
 OBJ := $(SRC:.c=.o)
OBJ := $(patsubst src/%,tmp/%,$(_OBJ))
#
#
.PHONY: $(BINEXE1) clean
$(BINEXE1): $(OBJ)
   $(CC1) -Iinc $(CFLAGS1) $(OBJ) $(LIBS) $(MAIN1) -o $@
   @[ -L $(EXE1) ] || (ln -s $@ .)


tmp/%.o: src/%.c $(MAIN1)
   $(CC) -c $(CFLAGS) -Iinc $< -o $@

clean:
   @rm -f $(EXE1) $(BINEXE1) $(OBJ)
```

Annex 6. Bash script to run the JRC demo source – file `test.sh`

```
lmdtest.x -e 1 1 1 In111.txt out.coer
lmdtest.x -d out.coer
```

GETTING IN TOUCH WITH THE EU

In person

All over the European Union there are hundreds of Europe Direct information centres. You can find the address of the centre nearest you at: https://europa.eu/european-union/contact_en

On the phone or by email

Europe Direct is a service that answers your questions about the European Union. You can contact this service:

- by freephone: 00 800 6 7 8 9 10 11 (certain operators may charge for these calls),

- at the following standard number: +32 22999696, or

- by electronic mail via: https://europa.eu/european-union/contact_en

FINDING INFORMATION ABOUT THE EU

Online

Information about the European Union in all the official languages of the EU is available on the Europa website at: https://europa.eu/european-union/index_en

EU publications

You can download or order free and priced EU publications from EU Bookshop at: https://publications.europa.eu/en/publications. Multiple copies of free publications may be obtained by contacting Europe Direct or your local information centre (see https://europa.eu/european-union/contact_en).

KJ-NA-30409-EN-N

The European Commission's science and knowledge service

Joint Research Centre

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub
ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

EU Science, Research and Innovation

EU Science Hub

Publications Office of the European Union

doi:10.2760/644570

ISBN 978-92-76-23523-1