V2X ASN.1 Python

Encode/Decode API

User's Guide

Objective Systems, Inc. April 2022

Introduction

The Objective Systems V2X Python API is a wrapper around the Objective Systems V2X C++ API. The API is implemented in Python and depends on the V2X C++ API shared library. It is compatible with Python 2.7 and Python 3.x. Due to the use of a shared library, the Python O/S and architecture must match the V2X API O/S and architecture (e.g. using Python for Windows 32-bit requires using the shared library from the Windows 32-bit V2X package).

As provided, the Python wrapper uses the v2xasn1_j2735_202007 and v2xasn1_etsi shared libraries. Refer to the V2X ASN.1 C++ User Guide for details on the specification versions supported by these libraries and, hence, by the Python wrapper. You can likely modify the Python source to use other versions of the J2735 or ETSI libraries, but we have not tested this.

The API provides simple function calls that can used to convert binary V2X messages encoded according to the Packed Encoding Rules (PER) to JSON and XML and vice versa. It supports SAE J2735 MessageFrame messages and the following ETSI ITS messages: CAM, DENM, SPATEM, MAPEM, IVIM, SREM, and SSEM.

This document contains reference documentation for the API as well as simple examples for calling the API to convert messages.

Changes since v74300

Starting in v74301, the supported revision of SAE J2735 was changed to 202007 (it had been 201603). This also means that the Python wrapper is now referring to a different shared library than previously.

Changes since v73400

Starting in v73401, the return type for the conversion methods (e.g. CAM.from_json), when there is an error, is now a tuple instead of an int. The tuple contains the integer error code and the error text. The conversion methods will no longer print out the error text to standard output.

Changes since v73300

The Python wrapper was formerly provided as a binary extension, dependent on Boost.Python, and supporting only Python 2.7. Since the wrapper is no longer provided as a binary extension, there are some slight changes in the requirements for using it (e.g., there is no longer a .pyd file to add to the path). However, the osys.v2x Python module is the same, so client code should not require any changes.

Package Contents

The V2X API installation has the following structure:

```
v2x_api_<version>
  +- doc
  +- python
  |  +- src
  |     +- osys
  +- sample
     +- python
```

<version> would be replaced with a 5-digit version number and <config> by a configuration identifier. The first 3 digits of the version number are the ASN1C version used to generate the API and the last two are a sequential number.

For example, v2x_api_v74002 would be the third version generated with the ASN1C v7.4.0 compiler.

The purpose and contents of the various subdirectories are as follows:

- python/src – Contains the folder hierarchy for the Python wrapper source code. This folder should be in your PYTHONPATH.
- doc – Contains this document.
- sample – Contains a sample Python program that illustrates how to use the API. Sample MessageFrame messages are also provided.

Getting Started

This package is delivered as a zipped archive (.zip) or a tar-gzipped archive (.tar.gz) that should be unpacked in the same directory structure as the already-installed V2X C++ API. The libraries needed to use the API are stored in the `lib` subdirectories.

The sample program shows how to use the API to convert from JSON and XML to hexadecimal text (or binary output) and vice versa. A script is provided (`conv.sh` or `conv.bat`) to show how to set the environment variables and to illustrate some command line options.

Windows

Windows users may use one of several methods to ensure that the DLLs are loaded on startup:

1. Place the `v2xasn1_j2735_202007.dll`, and `v2xasn1_etsi.dll` library files in a directory on the system-wide path. You don't necessarily need both shared libraries; you only need the one(s) corresponding to the set of specifications you are working with. Note that for Python 3.8, this does not include the PATH, but does include folders such as Windows\System32. (See https://docs.python.org/3/whatsnew/3.8.html#bpo-36085-whatsnew)
2. Set PATH. (Starting with Python 3.8, this does not work.) Update the path to include the directory in which the DLLs are loaded. From the command-line, use the set command. For example:

```
set PATH=%PATH%;c:\<v2x_root_dir>\debug\lib
```

3. Set environment variable `V2XDLLPATH` to the directory in which the DLLs are located.
4. Use `os.add_dll_directory` (new in Python 3.8) to add a folder to the DLL search path.

The `PYTHONPATH` variable will also need to be set to point to the directory that contains the osys package folder. For example:

```
set PYTHONPATH=%PYTHONPATH%;c:\<v2x_root_dir>\python\src
```

In the case of a limited binary library (which includes the evaluation edition), it may be necessary to assign another environment variable to allow the license file to be located. The `ACLICFILE` environment variable should be set to the full pathname to the `osyslic.txt` file that was provided with the product. For example, if you place the license file in the root directory of the installation, the following variable would need to be defined:

```
set ACLICFILE=c:\<v2x_root_dir>\osyslic.txt
```

Linux

Linux users may use one of the following methods to ensure that the shared libraries are loaded on startup:

1. Place the `libv2xasn1_j2735_202007.so`, and `libv2xasn1_etsi.so` library files in a directory searched by `ld`; a subdirectory of `/usr/lib` is a common location. Copying the files into these locations usually requires super-user privileges. You don't necessarily need both shared libraries; you only need the one(s) corresponding to the set of specifications you are working with.
2. Export the `LD_LIBRARY_PATH` environment variable prior to calling the application:

   ```
   export LD_LIBRARY_PATH=${HOME}/<v2x_root_dir>/debug/lib
   ```

3. Export the `V2XDLLPATH` environment variable to the directory in which the DLLs are located.

The `PYTHONPATH` variable will also need to be set to point to the directory that contains the osys package folder. For example:

```
export PYTHONPATH=${PYTHONPATH}:${HOME}/<v2x_root_dir>/python/src
```

As with the Windows kit, limited binary libraries will require setting the `ACLICFILE` environment variable. For example:

```
export ACLICFILE=$HOME/<v2x_root_dir>/osyslic.txt
```

Mac OS X

Mac OS X users may use one of the following methods to ensure that the shared libraries are loaded on startup:

1. Place the `libv2xasn1_j2735_202007.dylib`, and `libv2xasn1_etsi.dylib` library files in a directory that is searched for these files; the `/usr/lib` directory usually works. Copying the files into these locations usually requires super-user privileges. You don't necessarily need both shared libraries; you only need the one(s) corresponding to the set of specifications you are working with.
2. Export the `DYLD_LIBRARY_PATH` environment variable prior to calling the application:

   ```
   export DYLD_LIBRARY_PATH=${HOME}/<v2x_root_dir>/debug/lib
   ```

3. Export the `V2XDLLPATH` environment variable to the directory in which the DLLs are located.

The `PYTHONPATH` variable will also need to be set to point to the directory that contains the osys package folder. For example:

```
export PYTHONPATH=${PYTHONPATH}:${HOME}/<v2x_root_dir>/python/src
```

As with the Windows kit, limited binary libraries will require setting the `ACLICFILE` environment variable. For example:

```
export ACLICFILE=$HOME/<v2x_root_dir>/osyslic.txt
```

Using the Sample Program

The provided sample program, `v2x_conv.py`, illustrates how to convert messages from text formats (hexadecimal, JSON, and XML) to binary and vice versa. A batch file (`conv.bat`) or shell script (`conv.sh`) is included to help set the environment variables described above.

Help on how to use the application may be obtained by running the application from the command line with the `-h` switch:

```
Usage:

    v2x_conv.py <-j | -x | -b> [-o <output file>] [--type=<MessageFrame | CAM |
DENM>]
        <input file>

    Where:

        -j      Converts the <input file> to JSON, assuming it is binary.
                This is the default behavior.
        -x      Converts the <input file> to XML, assuming it is binary.
        -b      Converts the <input file> to binary; specify -j or -x to
                    convert from JSON or XML.  (JSON is default.)  If the
                    output file is not specified, an ASCII representation is
                    printed to standard output.
```

```
        -o      Outputs the content to <output file>.  Standard output is the
                    default output.
        -h      This help.

        --hex   Treats the input file as a hexadecimal text file, converting
                    it first to binary and then to the specified output
                    format.

        --type=<type> Treats the input message as one of the three listed PDU data
                    types in the V2X specifications:  MessageFrame, CAM, or
                    DENM.  By default, the MessageFrame PDU is assumed.

    For example:

        test.py -jb -o message.dat message.json

    or

        test.py -b -o message.dat message.json

    converts the input file message.json to a binary file message.dat,
    assuming it is a MessageFrame data type.  An equivalent set of options is

        test.py -j message.dat

    converts the input file message.dat to JSON, outputting it to standard
    output.

        test.py -x -o message.xml --type=CAM message.dat

    converts the input file message.dat to XML, outputting it to message.xml.
    The input PDU type is assumed to be a CAM message.
```

Sample data are provided with the program for the BasicSafetyMessage, CAMMessage, and DENMMessage types.

API Reference

The V2X Python classes are located inside of the `osys.v2x` package.

None of the classes is instantiable; instead they provide class methods for performing conversions to and from text and binary formats. Help text is available through the usual `help(classname)` functions in Python. The help text is reproduced here:

```
Help on module osys.v2x in osys:

NAME
    osys.v2x - osys.v2x

DESCRIPTION
    This module acts as a wrapper around the C++ V2X DLL.  It provides classes
    that enable conversion between binary and text representations of V2X messages.

    There are two C++ shared libraries that provide access to the V2X
functionality.
    The first is for SAE J2735 MessageFrame messages, and is named
```

```
    "v2xasn1_j2735_202007".  The second is for ETSI messages and is named
    "v2xasn1_etsi".
    This module is designed so that only the library being used is required to be
    present.

    The conversion functionality is implemented in one class per message type,
    in class methods.  The class methods are consistent across all types, see
    the documentation for _Message.  The class methods are:
        to_json
        from_json
        to_xml
        from_xml.
    The classes are:
        MessageFrame (J2735 202007)
        CAM
        DENM
        SPATEM
        MAPEM
        IVIM
        SREM
        SSEM

CLASSES
    _Message(builtins.object)
        CAM
        DENM
        IVIM
        MAPEM
        MessageFrame
        SPATEM
        SREM
        SSEM

    class CAM(_Message)
     |  The CAM class. ETSI EN 302 637-2.
     |
     |  The CAM class has no constructor: rather it offers four functions
     |  for converting messages from JSON or XML to a binary buffer and
     |  vice versa.
     |
     |  Method resolution order:
     |      CAM
     |      _Message
     |      builtins.object
     |
     |  Class methods inherited from _Message:
     |
     |  from_json(json_str, verbose=True) from builtins.type
     |      Returns the binary encoding (as a Python buffer) of an input
     |      JSON document or a tuple consisting of an error code (int) and an
     |      error message (str)).
     |
     |  from_xml(xml_str, verbose=True) from builtins.type
     |      Returns the binary encoding (as a Python buffer) of an input
     |      XML document or a tuple consisting of an error code (int) and an
     |      error message (str)).
     |
     |  to_json(dat, nbytes, verbose=True) from builtins.type
     |      Returns a Python buffer containing the JSON representation of
```

```
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class DENM(_Message)
 |  The DENM class. ETSI EN 302 637-3.
 |
 |  The DENM class has no constructor: rather it offers four functions
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      DENM
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      XML document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  to_json(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the JSON representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
class IVIM(_Message)
 |  The IVIM class. ETSI TS 103 301.
 |
 |  The IVIM class has no constructor: rather it offers four functions
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      IVIM
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      XML document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  to_json(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the JSON representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class MAPEM(_Message)
 |  The MAPEM class. ETSI TS 103 301.
 |
 |  The MAPEM class has no constructor: rather it offers four functions
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      MAPEM
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
```

```
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      XML document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  to_json(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the JSON representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class MessageFrame(_Message)
 |  The MessageFrame class. SAE J2735 202007.
 |
 |  The MessageFrame class has no constructor: rather it offers four functions
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      MessageFrame
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      XML document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  to_json(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the JSON representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
```

```
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class SPATEM(_Message)
 |  The SPATEM class. ETSI TS 103 301.
 |
 |  The SPATEM class has no constructor: rather it offers four functions
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      SPATEM
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      XML document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  to_json(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the JSON representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class SREM(_Message)
 |  The SREM class. ETSI TS 103 301.
 |
 |  The SREM class has no constructor: rather it offers four functions
```

```
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      SREM
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      XML document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  to_json(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the JSON representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  to_xml(dat, nbytes, verbose=True) from builtins.type
 |      Returns a Python buffer containing the XML representation of
 |      the input binary MessageFrame or a tuple consisting of an error code
 |      (int) and an error message (str)).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _Message:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class SSEM(_Message)
 |  The SSEM class. ETSI TS 103 301.
 |
 |  The SSEM class has no constructor: rather it offers four functions
 |  for converting messages from JSON or XML to a binary buffer and
 |  vice versa.
 |
 |  Method resolution order:
 |      SSEM
 |      _Message
 |      builtins.object
 |
 |  Class methods inherited from _Message:
 |
 |  from_json(json_str, verbose=True) from builtins.type
 |      Returns the binary encoding (as a Python buffer) of an input
 |      JSON document or a tuple consisting of an error code (int) and an
 |      error message (str)).
 |
 |  from_xml(xml_str, verbose=True) from builtins.type
```

```
|      Returns the binary encoding (as a Python buffer) of an input
|      XML document or a tuple consisting of an error code (int) and an
|      error message (str)).
|
|  to_json(dat, nbytes, verbose=True) from builtins.type
|      Returns a Python buffer containing the JSON representation of
|      the input binary MessageFrame or a tuple consisting of an error code
|      (int) and an error message (str)).
|
|  to_xml(dat, nbytes, verbose=True) from builtins.type
|      Returns a Python buffer containing the XML representation of
|      the input binary MessageFrame or a tuple consisting of an error code
|      (int) and an error message (str)).
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from _Message:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```

API Example: Converting JSON to Hex Text

Included in this package are sample data for a `BasicSafetyMessage` message. This is encoded as a `MessageFrame` PDU type, and so it corresponds to the `v2x.MessageFrame` class.

The following code might be used to convert the JSON message into a hexadecimal representation of the binary output:

```
from osys import v2x
import binascii

# import the JSON text from the input file
jstr = open('message.json', 'r').read()

# convert the JSON text into a Python buffer
data = v2x.MessageFrame.from_json(jstr)

# convert the buffer data into a hex string
hex = binascii.hexlify(data)

# finally, write it to a file
open('message.hex', 'wb').write(hex)
```

The conversion to hex is performed by the built-in `binascii` module. To convert XML to hexadecimal text simply requires changing the `from_json` method call to `from_xml`.

API Example: Converting Hex Text to JSON

We use the same sample data as above from the `BasicSafetyMessage`.

```
from osys import v2x
```

```
import binascii

# import the hexadecimal text from the input file
hstr = open('message.hex', 'r').read()

# convert the hexadecimal text to binary
data = binascii.unhexlify(hstr)

# convert the binary data to JSON
jstr = v2x.MessageFrame.to_json(data)

# write the JSON data to a file
open('message.json', 'w').write(jstr)
```

The conversion from hex is performed by the built-in `binascii` module. A conversion to XML simply requires changing the `to_json` method call to `to_xml`.