V2X ASN.1 Python

Encode/Decode API

User's Guide

Objective Systems, Inc. February 2019

Introduction

The Objective Systems V2X Python API is a wrapper around the Objective Systems V2X C++ API. The API is contained in a DLL (`.pyd` on Windows, `.so` on Linux) compatible with Python 2.7. The library depends on the Python and Boost::Python runtime DLLs, which are also provided.

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 the same message types as the C++ API (i.e., Dedicated Short Range Communications (DSRC) Message Set Dictionary as specified by SAE J2735, March 2016, and the European equivalents: CAM and DENM).

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

Package Contents

The V2X API installation has the following structure:

```
v2x_api_<version>
  +- doc
  +- debug
     +- lib
  +- release
     +- lib
  +- 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_dll_v72200 is be the initial version generated with the ASN1C v7.2.2 compiler.

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

- debug/lib – Contains the debug version of the V2X Python API and its supporting libraries (Boost::Python and Python).
- release/lib – Contains the release version of the V2X Python API and its supporting libraries (Boost::Python and Python).
- 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 two methods to ensure that the DLLs are loaded on startup:

1. Place the `osys.pyd`, `v2xasn1.dll`, `python27.dll`, and `boost_python27-vc140-mt-x64-1_67.dll` library files in a directory on the system-wide path.
2. 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
   ```

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

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

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 two methods to ensure that the shared libraries are loaded on startup:

1. Place the `osys.so`, `libv2xasn1.so`, `libpython27.so.1.0`, and `libboost_python.so.1.67.0` 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.

2. Export the LD_LIBRARY_PATH environment variable prior to calling the application:

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

The PYTHONPATH variable will also need to be set to point to the same directory that contains osys.so. For example:

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

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:

```
osys.v2x
|
+- CAM
   +- from_json
   +- from_xml
   +- to_json
   +- to_xml
|
+- DENM
   +- from_json
   +- from_xml
   +- to_json
   +- to_xml
|
+- MessageFrame
   |
   +- from_json
   +- from_xml
   +- to_json
   +- to_xml
```

None of the classes is instantiable; instead they provide static 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

FILE
    (built-in)
```

```
CLASSES
    Boost.Python.instance(__builtin__.object)
        CAM
        DENM
        MessageFrame

    class CAM(Boost.Python.instance)
     |  The CAM class.
     |
     |  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
     |      Boost.Python.instance
     |      __builtin__.object
     |
     |  Methods defined here:
     |
     |  __reduce__ = <unnamed Boost.Python function>(...)
     |
     |  ----------------------------------------------------------------------
     |  Static methods defined here:
     |
     |  from_json(...)
     |      from_json( (str)arg1) -> object :
     |          buffer = from_json(json_str)
     |
     |          Returns the binary encoding (as a Python buffer) of an input
     |          JSON document.
     |
     |          C++ signature :
     |              boost::python::api::object from_json(char const*)
     |
     |  from_xml(...)
     |      from_xml( (str)arg1) -> object :
     |          buffer = from_xml(xml_str)
     |
     |          Returns the binary encoding (as a Python buffer) of an input
     |          XML document.
     |
     |          C++ signature :
     |              boost::python::api::object from_xml(char const*)
     |
     |  to_json(...)
     |      to_json( (str)arg1, (int)arg2) -> object :
     |          json = to_json(data)
     |
     |          Returns a Python buffer containing the JSON representation of
     |          the input binary CAM.
     |
     |          C++ signature :
     |              boost::python::api::object to_json(char const*,unsigned long)
     |
     |  to_xml(...)
     |      to_xml( (str)arg1, (int)arg2) -> object :
     |          xml = to_json(data)
```

```
 |
 |          Returns a Python buffer containing the XML representation of
 |          the input binary CAM.
 |
 |          C++ signature :
 |              boost::python::api::object to_xml(char const*,unsigned long)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __init__ = <built-in function __init__>
 |      Raises an exception
 |      This class cannot be instantiated from Python
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Boost.Python.instance:
 |
 |  __dict__
 |
 |  __weakref__
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Boost.Python.instance:
 |
 |  __new__ = <built-in method __new__ of Boost.Python.class object>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T

class DENM(Boost.Python.instance)
 |  The DENM class.
 |
 |  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
 |      Boost.Python.instance
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  __reduce__ = <unnamed Boost.Python function>(...)
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  from_json(...)
 |      from_json( (str)arg1) -> object :
 |          buffer = from_json(json_str)
 |
 |          Returns the binary encoding (as a Python buffer) of an input
 |          JSON document.
 |
 |          C++ signature :
 |              boost::python::api::object from_json(char const*)
 |
 |  from_xml(...)
 |      from_xml( (str)arg1) -> object :
 |          buffer = from_xml(xml_str)
```

```
 |
 |          Returns the binary encoding (as a Python buffer) of an input
 |          XML document.
 |
 |          C++ signature :
 |              boost::python::api::object from_xml(char const*)
 |
 |  to_json(...)
 |      to_json( (str)arg1, (int)arg2) -> object :
 |          json = to_json(data)
 |
 |          Returns a Python buffer containing the JSON representation of
 |          the input binary DENM.
 |
 |          C++ signature :
 |              boost::python::api::object to_json(char const*,unsigned long)
 |
 |  to_xml(...)
 |      to_xml( (str)arg1, (int)arg2) -> object :
 |          xml = to_json(data)
 |
 |          Returns a Python buffer containing the XML representation of
 |          the input binary DENM.
 |
 |          C++ signature :
 |              boost::python::api::object to_xml(char const*,unsigned long)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __init__ = <built-in function __init__>
 |      Raises an exception
 |      This class cannot be instantiated from Python
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Boost.Python.instance:
 |
 |  __dict__
 |
 |  __weakref__
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Boost.Python.instance:
 |
 |  __new__ = <built-in method __new__ of Boost.Python.class object>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T

class MessageFrame(Boost.Python.instance)
 |  The MessageFrame class.
 |
 |  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
 |      Boost.Python.instance
 |      __builtin__.object
 |
```

```
|  Methods defined here:
|
|  __reduce__ = <unnamed Boost.Python function>(...)
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  from_json(...)
|      from_json( (str)arg1) -> object :
|          buffer = from_json(json_str)
|
|          Returns the binary encoding (as a Python buffer) of an input
|          JSON document.
|
|          C++ signature :
|              boost::python::api::object from_json(char const*)
|
|  from_xml(...)
|      from_xml( (str)arg1) -> object :
|          buffer = from_xml(xml_str)
|
|          Returns the binary encoding (as a Python buffer) of an input
|          XML document.
|
|          C++ signature :
|              boost::python::api::object from_xml(char const*)
|
|  to_json(...)
|      to_json( (str)arg1, (int)arg2) -> object :
|          json = to_json(data)
|
|          Returns a Python buffer containing the JSON representation of
|          the input binary MessageFrame.
|
|          C++ signature :
|              boost::python::api::object to_json(char const*,unsigned long)
|
|  to_xml(...)
|      to_xml( (str)arg1, (int)arg2) -> object :
|          xml = to_json(data)
|
|          Returns a Python buffer containing the XML representation of
|          the input binary MessageFrame.
|
|          C++ signature :
|              boost::python::api::object to_xml(char const*,unsigned long)
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __init__ = <built-in function __init__>
|      Raises an exception
|      This class cannot be instantiated from Python
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from Boost.Python.instance:
|
|  __dict__
|
```

```
 |  __weakref__
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Boost.Python.instance:
 |
 |  __new__ = <built-in method __new__ of Boost.Python.class object>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T
```

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`.