V2X ASN.1 Go Wrapper

Encode/Decode API

User's Guide

Objective Systems, Inc.   April 2022

Introduction

The Objective Systems V2X Go API is a wrapper around the Objective Systems V2X C++ API. The wrapper API is implemented in Go and depends on the V2X C++ API shared library.

As provided, the Go 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 Go wrapper. You can likely modify the Go source to use other versions of the J2735 or ETSI libraries, but we have not tested this.

The wrapper was developed using Go 1.14, as a Go module. The wrapper uses cgo to interface with the shared library. We expect the wrapper could be adapted to work with older versions of Go (not using modules), if needed.

Due to the use of a shared library, you will need to have the correct shared library for the platform you are building for.

Also, since cgo is used, you will need to have a gcc compiler available to the go build tool. You will need to use a version of gcc that will produce binaries compatible with the binaries produced by the go compiler. (Note: the Go compiler can do cross-compilation, so a full discussion of this topic is beyond the scope of this document. Golang's cgo documentation doesn't provide a lot of guidance here, but if you are doing cross-compilation, you may need to refer to it.) On Windows, using the Windows 64-bit version of Go and not doing cross-compilation, mingw64 (http://mingw-w64.org) is a suitable compiler; cygwin might or might not work.

The Go wrapper provides several global variables that implement a common interface for converting data. With a single method call, you can convert binary V2X messages, encoded according to the Packed Encoding Rules (PER), to JSON or XML, and vice versa. The wrapper 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 wrapper API as well as information on using the sample program.

Package Contents

The V2X API installation has the following structure:

```
v2x_api_<version>_<config>
     ├───golang
         ├───sample
         │   ├───data
         │   │   ├───BasicSafetyMessage
         │   │   ├───CAMMessage
         │   │   └───DENMMessage
         │   └───cmd
         │
```

```
│       ├───v2xconv
└───wrapper
    ├───c
    ├───common
    ├───etsi
    └───j2735
```

<version> would be replaced with a 5-digit version number. 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. <config> indicates the license control (or lack thereof) embedded in the shared libraries.

For example, v2x_api_v74002_lb would be the third version generated with the ASN1C v7.4.0 compiler.

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

- golang/wrapper – Contains the folder hierarchy for the Go wrapper source.
- golang/sample/cmd/v2xconv – Contains a sample Go program that illustrates how to use the API.
- golang/sample/data – Contains sample data that can be used with the sample program.

This user's guide is in the main doc subdirectory of the installation.

Getting Started

This section will describe how to build and run the sample program provided with the wrapper. You will need to follow similar steps to build your own program that uses the wrapper.

First, you will build the sample program. To do this:

1. Open a command shell and navigate to the golang/sample folder.
2. Make sure that gcc is in your PATH (see Introduction)
3. Run: go build ./...

This will produce golang/sample/v2conv(.exe).

To use v2xconv (or your own program using the wrapper), you will need to make sure the shared libraries (e.g. v2x_api_<version>/debug/lib) can be found by the system; details vary by operating system (see below). Having done that, simply run 'v2xconv' to see the usage text.

There are example Windows and Linux scripts in the sample folder (conv.bat and conv.bash) that set the necessary environment variables and call v2xconv with a variety of options.

The sample program code (sample/cmd/v2conv/sample.go) shows how to use the wrapper API to convert from JSON and XML to hexadecimal text (or binary output) and vice versa.

v2xconv Sample Usage

The sample program's usage text is self-explanatory:

```
Convert between text and binary formats using the following syntax.
   v2xconv (xml2bin | json2bin | bin2xml | bin2json) [-hex] [-verbose]
           (MessageName) in [out]

   The first argument indicates the kind of conversion to be performed. It is
   followed by any optional flags.

   The -hex flag indicates that hexadecimal should be used to represent the
   binary (UPER) message.

   The -verbose flag indicates diagnostic information should be written to
   standard output.

   MessageName is one of: "MessageFrame" (for J2735), or (for ETSI) "CAM",
   "DENM", "IVIM", "MAPEM", "SPATEM", "SREM", "SSEM".

   Output is sent to standard output unless a file name is given. If the
   conversion is to binary and no output is given, -hex is implied.
```

Windows

Windows users may use one of two 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. Whether you need both libraries will depend on whether you use both the obj-sys.com/golang/v2x-wrapper/j2735 and etsi packages in your program.
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
   ```

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 `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. Whether you need both

libraries will depend on whether you use both the obj-sys.com/golang/v2x-wrapper/j2735 and etsi packages in your program.

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

   ```
   export LD_LIBRARY_PATH=${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
```

API Reference

The V2X Go Wrapper API is written as a Go module with path obj-sys.com/golang/v2x-wraper. The sample program is part of a different Go module (obj-sys.com/golang/v2x-sample).

For each of the supported V2X message types, there is a variable of interface type common.Converter that can be used to perform the conversions for that message.

Here is the documentation for the common.Converter interface:

```
type Converter interface {
        FromJson(text string, verbose bool) (data []byte, err error)
        FromXml(text string, verbose bool) (data []byte, err error)
        ToJson(data []byte, verbose bool) (text string, err error)
        ToXml(data []byte, verbose bool) (text string, err error)
}
    Converter specifies the interface for converting between text and binary
    messages.

    The functions which convert from text to binary accept the text encoding in
    UTF-8 and return the binary data or an error.

    The functions which convert from binary to text accept the binary data and
    return the text encoding in UTF-8 or an error.

    When a conversion function is passed verbose=true, diagnostic information is
    printed to standard output.
```

Here is the documentation of the Converters provided by the etsi package:

```
VARIABLES

var (
        CAM    common.Converter = cAM{}
        DENM   common.Converter = dENM{}
        IVIM   common.Converter = iVIM{}
        MAPEM  common.Converter = mAPEM{}
        SPATEM common.Converter = sPATEM{}
        SREM   common.Converter = sREM{}
        SSEM   common.Converter = sSEM{}
)
```

```
    CAM et. al are Converters for the various ETSI messages.
```

And lastly, here is the documentation of the Converters provided by the j2735 package:

```
VARIABLES

var MessageFrame common.Converter = messageFrame{}
    MessageFrame is the Converter for J2735 MessageFrame messages.
```