SAE International

DSRC Implementation Guide

A guide to users of SAE J2735 message sets over DSRC

Revision History

No.	Date	By	Description
4	19-May-08	Caroline Michaels	Primary changes this time reflect FHWA comments and adding additional planned content and thoughts to the outline as content creation begins.
5	1-Aug-08	David Kelley	- Deliverable "Part A" only; all the front matter and most of Roy's architecture and VII text is in this section. Rest of text unchanged from Ver4. I have also added a line denoting where the "new" content is, to assist Noblis and others as they review each part. - Using the older SAE formatting, but will let Suz change this as she sees fit to some common format we can then use for the rest of it. - I have pasted in my own list of terms and acronyms. These will surely change as the work proceeds. Most of these come from the draft standard. - I prefer times roman 10pt as the base font for this work, and a mono-spaced font for the code sections. At present there is a lot of mixed fonts to clean up; need Suz to decide this for us.
6	6-Aug-08	David Kelley	Reviewed one more time for any changes resulting from suggestions by Noblis and others to confirm they were handled. A few new terms were added.
7	Aug-08	Roy Sumner	
8	19-Aug-08	David Kelley	- This version has DCK accepted text from Roy plus new comments from DCK to try and get things correct. Note that the order of the 3 applications/messages has been corrected. - Also has all current sections and text merged with it, and multiple header sections removed (Suz will restore when mature).
9	21-Aug-08	Roy Sumner	Document again merged to produce Rev 9 , ready for pre-press, over to Suz.
10	22-Aug-08	David Kelley	
11	27-Aug-08	Susan Chriss	Reformat and edit sections 1. & 2.
12	26-Aug-08	David Kelley	Added section 3.
13	8-Sep-08	Susan Chriss	Edited section 3.
14	11-Sep-08	Roy Sumner	Roy's edits to sections 1-3; ~~new text in brown~~, inconsequential edits not shown.
	15-Sep-08	Dave Kelley	Added section 6.
15	27-Sep-08	Susan Chriss	Edited section 6.
15b	29-Sep-08	DCK & Suz	Final edits.
16	24-Sep-08	Dave Kelley	Added section 4.
	13-Oct-08	Susan Chriss	Edited section 4.
	14-Oct-08	DCK & Suz	Final edits.
17	12-Nov-08	David Kelley	Address FHWA comments.
	15-Nov-08	Roy Sumner	Address FHWA comments.
18	18-Nov-08	Susan Chriss	Finalize changes; ~~new text in violet~~, inconsequential edits not shown.
19	21-Aug-09	David Kelley	Edits to section 3 (Application 1, BSM). Edits to section 4 (Application 2, RSA). Added section 5 (Application 3, PVM). Added section 7 (Extending Messages). Added annex A (FAQs).
20	16-Feb-10	David Kelley	Minor edits to sections 3, 4, 5. Added section 8 and annex B & C

CONTENTS

SECTION **PAGE**

SECTION **PAGE**

TABLE **PAGE**

FIGURE **PAGE**

Foreword

The intended audience of this guide is those persons charged with developing and deploying DSRC who will use the SAE J2735 message set in applications. Specifically, these are presumed to be technically savvy individuals familiar with the DSRC system, its protocol stacks, and some knowledge of programming messages. These persons will need additional information on the specific details of the DSRC message set itself, on some XML encoding aspects the message set uses, and on the practical use of ASN.1 technologies, all of which this guide provides. It is not presumed that these persons are experts in any of these areas, but that pragmatic examples will be sufficient for developers to build a message. It is intended that this document serve the needs of the *implementer* rather than the *manager* of the implementer.

1. DSRC Implementation Guide Overview

1.1 Purpose of the Guide

The purpose of this guide is to provide assistance and guidance to an implementer of SAE J2735 in conjunction with Vehicle Infrastructure Integration (VII) applications.

This includes helping the implementer to:

- Understand how to use the J2735 messages; i.e., how common messages are used in different application areas and with different vendor implementations and performance goals to achieve interoperability.
- Explain implementation options for sending J2735 message set elements, sometimes combining with ATIS and ITIS elements.
- Understand interface design issues.
- Understand how to implement *Dedicated Short Range Communications* (DSRC) message sharing and use the DSRC message framework; i.e., how a single message may support multiple applications using its content.

In addition, this guide provides:

- Practical advice and examples of how the messages are implemented consisting of fragments of working code employed to build three rudimentary working applications.
- Detailed examples of conforming messages, encoded in both ASN DER and in XML forms.
- Advice on guidelines for processing messages to avoid message handling problems in application-to-application sharing and in the communication stack. Specifically, issues of common message decoding and encoding libraries and proper layers (as in an OSI model) are addressed with guidance on how the developer may organize functionality between applications.

1.2 Scope of the Guide

This guide focuses on the implementation of the SAE J2735 message sets using the following three messages:

- Basic Safety Message
- Roadside Alert Message
- Probe Vehicle Message

This guide illustrates the use of these messages with one application for each message, as follows.

- The **Basic Safety Message** (BSM) is illustrated with the application Emergency Electronic Brake Lights. When a vehicle brakes hard, the Emergency Electronic Brake light application creates and sends a message to other vehicles using elements found in the BSM. This single message (often informally called the heartbeat message because it is constantly being exchanged with nearby vehicles) is used in many applications to keep track of surrounding vehicles and make decisions.
- The **Roadside Alert** (RSA) is the message used in the various traveler information applications, specifically in the Emergency Vehicle Alert message used to inform mobile users of nearby emergency operations. In this use, a message from either the public safety on-board units (OBUs) or the infrastructure is broadcast to the mobile user to provide information on traffic conditions. The types of data in the Roadside Alert message can

include information such as travel delays, incident and diversion data, construction messages, and other data that the traffic management center (TMC) may wish to deliver to drivers. In the Emergency Vehicle use of the message, an emphasis is placed on information transmitted from vehicles operating in the surrounding area. This guide describes the delivery and decoding of the message by the device in the vehicle. The mechanism for presenting the information to the driver is the responsibility of the designer of the mobile device.

- The **Probe Vehicle Message** (PVM) is used by multiple applications. Vehicles gather data on road and traffic conditions at intervals. This data is combined into messages and transmitted to the roadside units (RSUs) and from there onto TMCs and information service providers. This probe data provides information on traffic conditions, weather, and road surface conditions.

These three messages and their applications were chosen as being representative of the types of safety messages exchanges and message content that is typical in the overall message set; they are more mature than many of the other messages. The types of data elements (integers, enumerations, short strings) are representative of the content found and anticipated in the future editions of the standard. The encoding methods illustrated in this guide can be applied to future new messages which follow the same encoding rules.

This guide does not implement a real application or system. Rather, it uses a few practical aspects of each application (to obtain suitable data to encode) to illustrate how the messages are intended to be properly constructed and used.

It should be stressed that in developing this guide there is a focus on the messaging used, not the rest of a functional system and its needs. The principal intent is to develop and teach the implementation details of the messages, providing useful fragments of encoding so that the reader can confidently develop their own embodiment.

1.3 Organization of the Guide

Section 2. is a general overview of DSRC message handling issues including a few design considerations that extend beyond the message set itself. Also covered are the supported protocols as well as issues of message transport to and from the DSRC point of transmission. Requirements which the message set layer places on other layers (such as transmission priority and latency needs) are covered.

The next three sections are:

- Section 3. – Basic Safety Message
- Section 4. – Roadside Alert Message
- Section 5. – Probe Vehicle Message

For **each** of these three sections, the guide describes the application and the message in several subsections as follows:

- The description of the application
- The relevance to the VII National Systems Requirements (NSR)
- Communication parameters, including (where appropriate):
  - communication range requirements
  - message direction (senders and receivers)
  - broadcast interval
  - priority assignment criteria

- How to use the message
- Representative encoding of message
- Examples of well-formed messages
- Relevance to ATIS or ITIS message
- Interface design issues/considerations
- Message sharing issues/considerations

The first of these two subsections describe how the selected application fits into the overall plan of the matrix of defined VII applications and the national architecture, emphasizing how the messages support the application, rather than the details of the application itself.

The next subsection (parameters) provides an overview of many important functional details which any real time application must support to be successful, although many of these do not directly impact the message encoding or reside in other layers of the overall DSRC protocol.

The next three subsections examine how to build working messages with multiple examples provided. In each of these subsections, a simple working tool for that message is built up to assist developers in creating representative and correct message encodings of their own.

The final three subsections touch on various resource sharing design considerations that any application needs to deal with in a shared environment (multiple applications in the same hardware).

This same format is followed for **each** for the three applications (sections 3., 4., and 5.); although some content is not repeated in each new section (i.e., the application text builds on each other and should be read in order). It is anticipated that in any future editions of the guide, this format will be followed as well.[1]

Section 6. deals with general information about ASN.1, and is provided to educate the reader about many of the issues of using ASN.1 with the Distinguished Encoding Rules (DER) encoding style.

Section 7. addresses several issues concerned with extending the message set to add new content to it and provide recommend solutions to this when needed.

Section 8. is dedicated to discussing the utility tool and the dialogs developed for each application. Some users are expected to use this tool simply to understand the messages, while other users may want to take the code examples provided and use it as a basis to develop their own tools. This section addresses use of the tool and provides an overview of the provided code.

[1] Although not funded or proposed to FHWA at this time, a guide section dealing with the SPAT message and its use in signalized intersections is foreseen.

1.4 Guide Conventions

Proper language code names of data elements, date frames, or messages are given in a mono-spaced font as in this example: the proper name of the Basic Safety Message is `BasicSafetyMessage` with the run together CamelCase style of naming as shown.

Such names may represent the ASN.1 or XML name for the element; typically this is the same string. However, a few exceptions to this rule can be found, typically in other areas of *Intelligent Transportation Systems* (ITS) standards which DSRC may reuse. Unless there is a language conflict issue,[2] these names tend to be used in the code examples for variables that represent them.

Proper data dictionary names of data elements, date frames, or message defined in the DSRC standard are always preceded with a short acronym to denote their type. For data structures defined by the SAE, these include:

- MSG_ for messages
- DF_ for data frames (structures with complex data content)
- DE_ for atomic data elements.

For example, the proper data dictionary name of the BSM is *MSG_BasicSafetyMessage* and are shown in italic type to emphasize that it is a proper name. This style of naming is generally found throughout ITS standards; however, a few exceptions to this rule can be found, typically in other areas of ITS standards in which DSRC may reuse an entry.

In general, data concepts defined in the DSRC standard are a named with a CamelCase style of naming with run-together names. This is also followed whenever possible in the variables defined in the code fragment example given here. For the fragments of code given in the C language, a Hungarian style has been used, and common Microsoft data naming conversions are often used by default.[3]

The proper filenames of source code listing are set in an italic type as in *stdio.h* when mentioned in the text.

Fragments of code listings, in various languages including C, VB, ASN, and XML, are given in a mono-spaced font as in the example below.

```
ReferencePoint ::= SEQUENCE {
   lat         Latitude,           -- 4 bytes (1/8th micro degree)
   long        Longitude,          -- 4 bytes
   elev        Elevation OPTIONAL, -- 3 bytes
   ...
   }
```

Examples of hexadecimal based values are preceded by `0x` as in `0x1F` for the base ten value of 31.

Examples of binary based values are preceded by `0Bx` as in `0Bx0101` for the base ten value of 5.

Electric and online copies of the standard and this document make extensive use of hinting and metadata which appears as highlighted pop-up text when the cursor hovers over it. This information is provided to assist those with visual disabilities in reading the content of the document.

[2] Underscores and dashes in names present problems for some languages. On a few occasions, the automatic mangling of these characters in a schema element name has resulted in different data concepts colliding in the target deployment. The source code presented here is free from this defect, but deployments should be aware it can occur.

[3] The general style of coding established by Microsoft and used in most of their published examples as well as countless third party textbooks and codes fragments and other guides is used here. No attempt is made to convert differing styles in this regard.

1.5 References

Below are the standards and other supporting documents used by reference.

The DSRC Message Set Standard, SAE J2735, can be obtained from:
http://www.sae.org/technical/standards/J2735_200911

The XML schema of the adopted DSRC Message Set Standard, SAE J2735, can be found at:
http://www.sae.org/standardsdev/dsrc/

The ASN.1 code listing of the adopted DSRC Message Set Standard, SAE J2735, can be found at:
http://www.sae.org/standardsdev/dsrc/DSRC_R36_Source.ASN

The XML schema of the adopted ITIS Code Standard, SAE J2340, Part 2, 4th edition, can be found at: http://www.sae.org/standardsdev/dsrc/DSRC-Adopted-02-00-36.xml

The ASN.1 code listing of the adopted ITIS Code Standard, SAE J2340, Part 2, 4th edition, can be found at: http://www.ITSware.net/itsschemas/ITIS/ITIS-04-00-03/ITIS_04_00_03_Source.ASN.txt

Vehicle Infrastructure Integration (VII) National System Requirements Version 1.3.1 April 2008 Draft. For the current status of the VII program, see:
www.its.dot.gov/vii/ and www.vehicle-infrastructure.org/

For links to other ITS standards, see:
http://www.standards.its.dot.gov/default.asp or the issuing Standards Development Organization (SDO).

The SAE Advanced Traveler Information System (ATIS) User Guide (a set of multiple volumes) can be downloaded from:
http://serv1.itsware.net/itsschemas/ATIS%20Guide/AGuidetotheATISGuides/
Extensive examples of the proper use of International Traveler Information Systems (ITIS) codes can be found in the examples in the ATIS Users Guide cited above.

The textbook TCP/IP Explained by Philip Miller, published by Digital Press, ISDN 1-55558-166-8. While this book is now a bit dated (published in 1997, it contains no information on the newer wireless protocols such as IEEE 802.11abc standard), it is considered one of the best references available for understanding the basics of TCP/IP communications.

A variety of ASN.1 resources including textbooks, various utility tools, and downloadable aids is maintained by French Telecom at:
http://asn1.elibel.tm.fr/en/resources/index.htm and http://asn1.elibel.tm.fr/en/index.htm
Among the contents of this site is John Larmouth's tutorial on ASN.1 tag rules, which has direct application for the ASN.1 in this guide; see:
http://www.larmouth.demon.co.uk/tutorials/tagging/

A web page of freely available textbooks and training aids regarding the language ASN.1 is maintained by OSS Nokalva at:
http://www.oss.com/asn1/booksintro.html
This site contains freely downloadable copies of John Larmouth's textbook ASN.1 Complete and of the Olivier Dubuisson textbook ASN.1 – Communications Between Heterogeneous Systems.

The specific rules for ASN and its encoding into DER can be found in the following ITU-T documents:
The ASN standard itself: http://www.itu.int/ITU-T/studygroups/com17/languages/X694.pdf
Various assignments used: http://www.itu.int/ITU-T/asn1/database/itu-t/x/x892/2005/index.html
Distinguished Encoding Rules: http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf

1.6 Definitions

For the purposes of this guide, the following definitions apply.

Term	Definition
Advisory Agent	A term used in the VII NSR documents. A System User that can both provide VII advisory messages for broadcast through the VII System or subscribe to receive VII advisory messages broadcast through the VII System.
Advisory Provider	A term used in the VII NSR documents. A System User that can provide VII advisory messages for broadcast through the VII System.
Advisory Subscriber	A term used in the VII NSR documents. A System User that can subscribe to receive VII advisory messages broadcast through the VII System.
airlink	A radio frequency communication interface, such as that defined by WAVE.
application class identifier (ACID)	A code that identifies a class of application (as defined in IEEE Std 1609), a term no longer used.
application context mark (ACM)	A code identifying a specific instance of an application (as defined in IEEE Std 1609 documents), a term no longer used.
application resource sharing	A software routine that can be used by multiple applications. For example, a routine that reads the vehicle location is used by multiple applications.
application-specific data dictionary	A data dictionary specific to a particular implementation of an ITS application. Local deployments which use DSRC (or other message sets) may often select a subset of the defined messages meeting their specific needs and create an application-specific data dictionary for that deployment.
base type	In the context of this standard, a base type is the primitive type (in either the ASN or XML language) from which a data concept is derived. For example, a type defined by the standard might be *DE_myNumber* which in turn was derived from the integer object. The precise encoding process used for named types defined by the standard varies depending on the base type from which they come. A further complication is that the type or structure used to represent the data concept when in memory (which may vary based on the programming language used) is not always an obvious or a one to one match. Details of this issue are discussed on a case by case basis in section 6.
Basic Encoding Rules (BER)	Quoting from a well known web reference (with some editing): The Basic Encoding Rules were the original rules laid out by ASN.1 for encoding abstract information into a concrete data stream. BER is further divided into two *flavors*, that of CER and DER. DER is the encoding style used in the DSRC message set over the WSM media.
Berkeley sockets (slang)	In the context of this standard, the style of API which is used to handle the layers of communications protocols below the application layer (which handles the DSRC message set defined by SAE J2735). A well known web reference states: The Berkeley socket API forms the de facto standard abstraction for network sockets. Most other programming languages use an interface similar to the C API.
byte type encoding	A type of information encoding where units of information are handled in modular increments of 8 bits.

Term	Definition
Canonical Encoding Rules (CER)	Quoting from a well known web reference (with some editing): Canonical Encoding Rules are a restricted variant of the ASN Basic Encoding Rules for producing unequivocal transfer syntax for data structures described by ASN.1. Whereas BER gives choices as to how data values may be encoded, CER and DER select just one encoding from those allowed by the BER, eliminating all of the options. They are useful when the encodings must be preserved, e.g., in security exchanges. CER and DER differ in the set of restrictions that they place on the encoder. The basic difference between CER and DER is that DER uses definitive length form and CER uses indefinite length form in some precisely defined cases. That is, DER always has leading length information, while CER uses the end-of-contents octet instead of providing the length of the encoded data. The CER style is not used in the DSRC message set work; all ASN messages are encoded with DER encoding.
control channel (CCH)	The radio channel of those defined in IEEE 802.11p used for exchange of management data and WAVE Short Messages.
data	Representations of static or dynamic entities in a formalized manner suitable for communication, interpretation, or processing by humans or by machines.
data concept	Any of a group of data dictionary structures defined in this standard (e.g., data element, data element concept, entity type, property, value domain, data frame, or message) referring to abstractions or things in the natural world that can be identified with explicit boundaries and meaning and whose properties and behavior all follow the same rules.
data consumer	Any entity in the ITS environment which consumes data from others.
data dictionary	An information technology for documenting, storing and retrieving the syntactical form (i.e., representational form) and some usage semantics of data elements and other data concepts. The major message sets of ITS, of which DSRC is but one, are kept and represented in a data dictionary.
data element (DE)	A syntactically formal representation of some single unit of information of interest (such as a fact, proposition, observation, etc.) with a singular instance value at any point in time, about some entity of interest (e.g., a person, place, process, property, object, concept, association, state, event). A data element is considered indivisible.
data frame (DF)	(Formerly Data Structure, which appears in the early ITS efforts, is now more commonly called a Data Frame; the definition and meaning, which follows, remains the same.) Any construct used to represent the contents of a data dictionary. From a computer science perspective, data frames are viewed as logical groupings of other data frames and of data elements to describe *structures* or parts of messages used in this and other standards. A data frame is a collection of one or more other data concepts in a known ordering. These data concepts may be simple (data elements) or complex (data frames).
data plane	The communication protocols defined to carry application and management data across the communications medium.
Data Provider	A term used in the National VII architecture documents. A System User that can provide raw data to the VII System.

Term	Definition
data registry	An advanced data dictionary that contains not only data about data elements in terms of their names, representational forms and usage in applications, but also substantial data about the semantics or meaning associated with the data elements as concepts that describe or provide information about real or abstract entities. A data registry may contain abstract data concepts that do not get directly represented as data elements in any application system, but which help in information interchange and reuse both from the perspective of human users and for machine interpretation of data elements. Within the ITS industry, there is a data registry established and run by the IEEE which contains the contents of this standard. SAE and the ATIS committee have also developed tools to access and use the data found in the registry as an aid to deployments.
data structure	Any construct (including data elements, data frames, and other data concepts) used to represent the contents of a data dictionary.
Data Subscriber	A term used in the National VII architecture documents. A System User that can subscribe to receive VII probe data collected through the VII System.
data type	A classification of the collection of letters, digits, and/or symbols used to encode values of a data element based upon the operations that can be performed on the data element; e.g., real, integer, character string, Boolean, bitstring, etc.
dialog	A sequence of two or more messages which are exchanged in a known sequence and format (typically of a request followed by one or more replies), which are considered a bound transactional exchange between the parties.
Distinguished Encoding Rules (DER)	Quoting from a well known web reference (with some editing): A term found in ASN to refer to a specific type of data encoding. DER is a method for encoding an ASN data object; it is the method of encoding used by the DSRC message set. The Distinguished Encoding Rules of ASN.1 is an International Standard drawn from the constraints placed on BER encodings by X.509. DER encodings are valid BER encodings. DER is the same thing as BER with all but one sender's options removed. For example, in BER a Boolean value of true can be encoded in 255 ways, while in DER there is only one way to encode a Boolean value of true. Loosely put, DER can be seen as a canonical form of BER (see also Canonical Encoding Rules). DER encoding is preferred (over the alterative CER) when the length of data content is always known, as is found in DSRC messages. In the DSRC message set work, all ASN messages are encoded with DER encoding.
DSRC equipment	In the context of this standard, DSRC equipment refers to any device which implements any portion of the DSRC message set described by SAE J2735. It should be noted that other application message set content (defined in other bodies of work) is presumed to co-exist with these messages.
encounter	In the context of this standard, an encounter is an exchange of messages between two or more DSRC equipped devices (OBUs or RSUs) lasting for a brief period of time.
entity	Anything of interest (such as a person, place, process, property, object, concept, association, state, event, etc.) within a given domain of discourse (in this case within the ITS domain of discourse).
entity type	An abstract type of structure defined in the ITS data register but no longer used. There are no entity types defined in this standard.
EtherType	The Ethernet Type field, as defined in RFC 1042, used to indicate the higher layer protocol above Logical Link Control.

Term	Definition
eXtensible Markup Language (XML)	A common method of exchanging messages made up of tags and values organized in a data structure and typically transported over common Internet formats such as HTTP. XML has a growing number of supporters due to its ability to be implemented in the types of heterogeneous systems often found in ITS deployments. It is possible to express and exchange the DSRC message sets using this method.
formats of sending	In the context of this standard, a format of sending refers to the precise choices of encoding format used, (including any inner encoded data elements represented as BLOB data), and the precise protocol used (typically the WSM message encoding format) as well as any required data that needs to be provided to the lower layers of protocol stack from the application layer. All of these aspects are specified in detail in the SAE J2735 message set document itself, and practical examples of their use are given in the guide.
functional area data dictionary (FADD)	A data dictionary that is intended to standardize data element syntax, and semantics, within and among application areas within the same functional area. This DSRC standard is a FADD.
GID layer	A set of physical geometry descriptions and roadway attributes sent in the map message to support the needs of OBUs for complex descriptions. The information is provided in logical groupings such as intersection descriptions, but may be integrated and stored in the vehicle in various layers as defined by DSRC algorithm developers.
intelligent transportation systems (ITS)	Systems that apply modern technology to transportation problems. Another appropriate meaning of the ITS acronym is integrated transportation systems, which stressed that ITS systems often integrate components and users from many domains, both public and private.
International Traveler Information System (ITIS)	International Traveler Information System is the term commonly associated with the standard for incident phrases developed by the SAE ATIS committee in conjunction with ITE TMDD and other standards. This work contains a wide variety of standard phrases to describe incidents and is expected to be used throughout the ITS industry. The codes found there can be used for sorting and classifying types of incident events, as well as creating uniform human readable phrases. In the capacity of classifying incident types, ITIS phrases are used in many areas. ITIS phrases can also be freely mixed with text and used to describe many incidents.
interoperability	The ability to share information between heterogeneous applications and systems.
link	A service channel being used in support of application data transfer needs.
listener (slang)	In the context of this standard, a listener is a element of code implemented on a DSRC device which listens on one or more known ports (or sockets) on some type of network card, and which is responsible for processing incoming messages when they appear on that port. It is a common practice for listener devices to provide the first stage of processing all incoming messages on its assigned port and also to handle the distribution of messages to more than one receiving application when needed, much like the functionally ascribed to a *message dispatcher* in some DSRC literature.
management plane	The collection of functions performed in support of the communication system operation, but not directly involved in passing application data.
message	A well structured set of data elements and data frame that can be sent as a unit between devices to convey some semantic meaning in the context of the applications about which this standard deals.
message set	A collection of messages based on the ITS functional area they pertain to.
metadata	Data that defines and describes other data.

Term	Definition
networking services	The collection of management plane and data plane functions at the network layer and transport layer, supporting WAVE communications.
node configuration data	*Definition to be refined by committee.* When a map of an intersection is represented, a node configuration data value provides key information regarding how the data values found in the map are to be understood.
notification	An indication of an event of interest, sent to an application.
OBU to vehicle host interface (OVHI)	Interface on the OBU offering access to WAVE capabilities by other vehicle-based devices.
on-board unit (OBU)	An on-board unit is a vehicle mounted DSRC device used to transmit and receive a variety of message traffic to and from other DSRC devices (other OBUs and RSUs). Among the message types and applications supported by this process are vehicle safety messages used to exchange information on each vehicle's dynamic movements for coordination and safety.
operations	One of three modes, or states, of operation known as Registration, Initialization, and Operations which DSRC systems operate in. In the Operations mode, a link has been established: the link has an open socket with which it can conduct operations in the same manner as with any other 802.11 communications session; the lower layers manage the switching between the control channel and the service channel. When the radio has switched to another channel, it would appear to the application as a temporary loss of communications.
Packed Encoding Rules (PER)	Quoting from a well known web reference (with some editing): Packed Encoding Rules are ASN.1 encoding rules for producing a compact transfer syntax for data structures described in ASN.1, defined in 1994. PER provides a much more compact encoding than BER-DER or BER-CER, at a cost of greater decoder complexity. It tries to represent the data units using the minimum number of bits. The compactness requires that the decoder knows the complete abstract syntax of the data structure to be decoded. The PER style is not used in the DSRC message set work; all ASN messages are encoded with DER encoding.
provider service table (PST)	The collection of data describing the applications that are registered with a WAVE device.
registration	One of three modes, or states, of operation known as Registration, Initialization, and Operations which DSRC systems operate in. The Registration mode is the process by which critical parameters pertaining to the device and applications using it are entered into the device's Management Information Base (MIB). Registration must be completed before a DSRC device can be ready for operations. The registration process is defined in IEEE P1609.3 and is controlled by the WME.
roadside unit (RSU)	A roadside unit is a DSRC device used to transmit to, and receive from, DSRC equipped moving vehicles (OBUs). The RSU transmits from a fixed position on the roadside (which may in fact be a permanent installation or from temporary equipment brought on-site for a period of time associated with an incident, road construction, or other event). RSUs have the ability to transmit signals with greater power than OBUs and may have TCP/IP connectivity to other nodes or the Internet.
service channel	Secondary channels used for application-specific information exchanges.
service table	A data store containing the pertinent information about applications available through the WAVE device.
stability control	A system which operates to prevent a car from sliding sideways under dynamic driving conditions.

Term	Definition
station	Any device that contains an IEEE 802.11 conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium. An RSU and OBU are stations.
syntax	The structure of expressions in a language, and the rules governing the structure of a language.
Transaction Agent	A term used in the National VII architecture documents. A System User that can send and receive application messages through the VII System to or from another Transaction Agent.
transactions	Bi-directional data exchanges between devices (RSUs and OBUs).
type definition	A term of use found in ASN to refer to data structures defined by the ASN being used. A type definition (such as `MyDataItem`) is defined and made up of one or more other types, composed of either other type definitions or of primitive types such as integers, strings, sequences, etc. All the definitions found in the DSRC message set standard can be traced back to primitive building blocks defined in the ASN language. This becomes important when the ASN is encoded into a format for transmission over the wire. In the DSRC message set standard, the style of encoding is ASN DER, and details of encoding each primitive type used can be found in section 6.
value domain	A well known range of values, or terminology, or enumeration that many be referenced as an abstract type in the ITS data register, but no longer used. There are very many value domains used in ITS standards.
vehicle host	A device connecting to the WAVE system through the OBU vehicle host interface.
vehicle type	In the context of this standard, the vehicle type is a data element used to define overall gross size and mass of a vehicle. Observe that this definition differs from the (multiple other) vehicle types defined elsewhere in other standards used in the ITS.
WAVE device	A device that contains a WAVE-conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium (see IEEE 802.11 and IEEE 1609.4).
WAVE management entity (WME)	The set of management functions, as defined in IEEE 1609 Std documents, required to provide WAVE networking services.
WAVE service information element (WSIE)	A collection of configuration data transmitted by either OBU or RSU, which includes the PST, and in the case of the RSU the WAVE Routing Advertisement, as well as security credentials.
wireline	Connected via a traditional communications interface; not the wireless interface specified here.
XML	see eXtensible Markup Language

1.7 Abbreviations and Acronyms

The industry terms, abbreviations and acronyms cited below are used in this guide unless specifically cited otherwise.

Acronym	Expansion
AAMVA	American Association of Motor Vehicle Administrators
AASHTO	American Association of State Highway and Transportation Officials.
ABS	Anti-lock Braking System
ACID	Application Class IDentifier

Acronym	Expansion
ACM	Application Context Mark
AID	Application IDentification
AMDS	Advisory Message Distribution Service
ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN	Abstract Syntax Notation
ASTM	American Society for Testing and Materials
ATIS	Advanced Traveler Information Systems
ATMS	Advanced Transportation Management System
BER	Basic Encoding Rules
BLOB	Binary Large OBject
BSM	Basic Safety Message
C2C	Center to Center
CALM	Continuous communications Air interface for Long and Medium range
CCC	Configuration Control Committee
CCH	Control Channel
CER	Canonical Encoding Rules
COTS	Commercial Off-The-Shelf
CRC	Cyclic Redundancy Code
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules
DF	Data Frame
DSRC	Dedicated Short Range Communications
ESS	Environmental Sensor Station
FADD	Functional Area Data Dictionary
FHWA	Federal HighWay Administration
FOT	Field Operation Test
FPV	Full Position Vector
GID	Geometric Intersection Description
GMT	Greenwich Mean Time
GNU	Gnu's Not Unix
GPS	Global Positioning System
HMI	Human-Machine Interface
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IM	Incident Management or inter-modal
IP	Internet Protocol
IPv6	Internet Protocol version 6
ISO	International Standards Organization
ITE	Institute of Transportation Engineers
ITIS	International Traveler Information Systems

Acronym	Expansion
ITS	Intelligent Transportation Systems
ITU	International Telecommunications Union
J2735	SAE standard titled: DSRC Message Set Dictionary
LLC	Logical Link Control
LRMS	Location Referencing Message Specification
LSB	Least Significant Bit
MAC	Medium Access Control
MEDS	Map Element Distribution Service
MIB	Management Information Base
MIL	Malfunction Indicator Light (check engine light)
MSB	Most Significant Bit
MUTCD	Manual on Uniform Traffic Control Devices
NAP	Network Access Point
NEMA	National Electrical Manufacturers Association
NIC	Network Interface Card
NSR	National System Requirements
NTCIP	National Transportation Communications for ITS Protocol
NTRIP	Networked Transport of RTCM via Internet Protocol
OBU	On-Board Unit
OER	Octet Encoding Rules
OID	Object IDentifier
OSI	Open Systems Interconnection
OVHI	OBU to Vehicle Host Interface
PDA	Personal Digital Assistant
PDS	Probe Data Service
PDU	Protocol Data Unit
PER	Packed Encoding Rules
PHY	Physical Layer
PKI	Public Key Infrastructure
PMUS	Private Mobile User Segment
POSIX	Portable Operating System Interface
PSA	Provider Service Announcement
PSC	Provider Service Context
PSMU	Public Service Mobile User
PSN	Probe Segment Number
PST	Provider Service Table
PVD	Probe Vehicle Data
PVM	Probe Vehicle Message
R2V	Roadside to Vehicle (communications)
RFC	Request For Comments
RSA	RoadSide Alert
RSE	Roadside Equipment

Acronym	Expansion
RSU	RoadSide Unit
RTCM	Radio Technical Commission for Maritime services
RTK	Real Time Kinematics
SAE	Society of Automotive Engineers
SAP	Service Access Point
SC-104	Special Committee 104 of the RTCM
SCH	Service Channel
SDN	Service Delivery Node
SDO	Standards Development Organization
SNMP	Simple Network Management Protocol
SPAT	Signal Phase And Timing
SRS	Safety Restraint System
TC	Traction Control
TC 204	Technical Committee 204 (Intelligent Transportation Systems)
TCIP	Transit Communications Interface Profiles
TCP	Transmission Control Protocol
TCS	Traction Control System
T-L-V	Type-Length-Value (or Tag-Length-Value)
TMC	Traffic Management Center
TMDD	Traffic Management Data Dictionary
UDP	User Datagram Protocol
USDOT	United States Department of Transportation
UTC	Universal Coordinated Time (or Universal Time, Coordinated; see *http://www.aldridge.com/client_serv/resources/utc.html*)
V2R	Vehicle to Roadside (communications)
V2V	Vehicle to Vehicle (communications)
VII	Vehicle Infrastructure Integration
VMS	Variable Message Sign
WAVE	Wireless Access in Vehicular Environments
WME	WAVE Management Entity
WSA	Wave Service Announcement
WSDL	Web Services Description Language
WSIE	WAVE Service Information Element
WSM	WAVE Short Message
WSMP	WSM Protocol
XER	XML Encoded Rules
XML	eXtensible Markup Language
XSD	XML Schema Datatypes

2. DSRC Concepts

2.1 DSRC Overview

This guide is not intended to provide a definitive description of Vehicle Infrastructure Integration; for that the reader is directed to the National System Requirements and the various other references described in this guide.

The USDOT describes the VII as follows:

> The VII Initiative is a cooperative effort among *U.S. Department of Transportation*, state and local governments, the automobile industry, and other partners to support development of an information infrastructure for ongoing real time data communications with and among vehicles to enable a number of safety, mobility, and commercial applications.

VII pictures an implemented VII network that will enable travelers to access traffic conditions and routing information for multiple modes of travel, receive warnings about imminent hazards, and conduct commercial transactions within their vehicles. Transportation agencies will have access to data needed to better manage traffic operations, support planning, and more efficiently manage maintenance services. Primary applications include:

- Warning drivers of unsafe conditions or imminent collisions
- Warning drivers if they are about to run off the road or speed around a curve too fast
- Informing system operators of real time congestion, weather conditions and incidents
- Electronic payment capabilities
- Providing operators with information on corridor capacity for real time management, planning and provision of corridor-wide advisories to drivers.

The goals of the VII initiative are to assess the value, determine the technical feasibility, economic viability, and social acceptance, and conduct research needed to support development of a nationwide VII network.

The original concept for VII involves a nationwide network that links and manages a series of roadside units. Although this concept is now under evaluation by the USDOT, this guide assumes that the network approach and architecture as embodied in the National System Requirements (NSR) will be implemented. The messages within SAE J2735 that form the basis for this guide are essentially independent of the transmission medium which is currently under review. For example, if there are data elements that are captured by the vehicle and transmitted to a mobility application at a traffic management center (TMC), the route they have taken does not affect the utility of the data provided it arrives in a timely manner.

The VII architecture and the subsequent NSR is structured to provide connectivity between the infrastructure and the vehicles and between the vehicles themselves. It is not prescriptive of how the data is used, how the information is displayed, or indeed how it is collected. It is essentially a conduit that provides data between the vehicles and the various institutions that want to connect to them. The figure below (Figure 1) illustrates the VII System.

The system does not store data, nor is it intended to turn data into information for users. These processes take place in external applications, either mobile or stationary, that interface with the system. It is these interfaces where the data is defined by SAE J2735 and it is these data that are the subject of this guide.

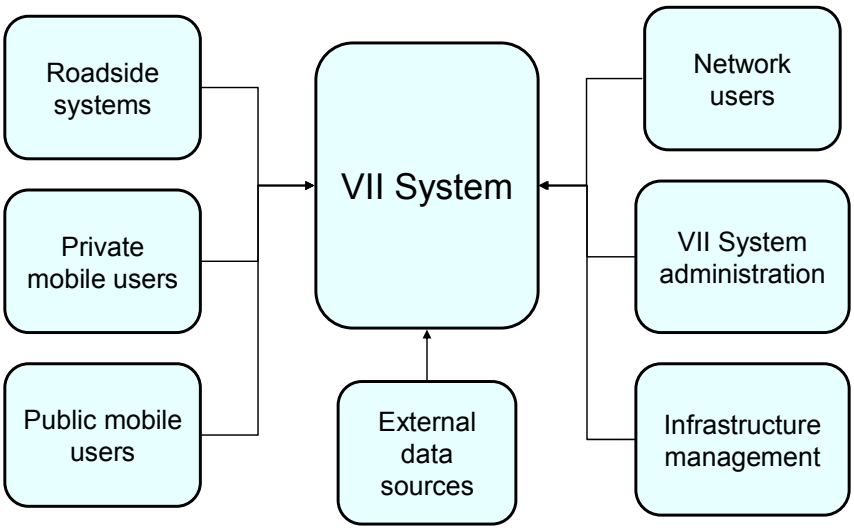

Figure 1 – Vehicle Infrastructure Integration System

The elements of the figure are described below.

The **VII System** is a wide area managed application that provides connectivity between the various users. The institutions connected to the system are varied and include all levels of government, the automakers, commercial companies and academia. The VII System is configured within the NSR with a variety of agents that perform the principal functions. These are:

- Advisory Agent provides advisory messages for broadcast through the System or subscribe to receive VII advisory messages broadcast through the System
- Advisory Provider provides VII advisory messages for broadcast through the System
- Advisory Subscriber subscribes to receive advisory messages broadcast through the System
- Data Provider can provides raw data to the VII System
- Data Subscriber subscribes to receive VII probe data collected through the VII System
- Transaction Agent sends and receives application messages through the System to or from another Transaction Agent

The **Private Mobile Users** are the devices that are present on the transportation network. These can include units built into vehicle and personal digitals assistants (PDAs) such as phones and navigation devices.

The **Public Mobile Users** may be similar devices as those that are private but they are configured differently in that privacy is not maintained. These devices would be installed in such vehicles as ambulances and fire trucks.

The **Roadside Systems** are devices such as signal controllers and ramp controls. Such systems form part of the VII System as they can gather data from the System for use in adopting local control strategies, and in addition provide support such as updates of security certificates.

The **Infrastructure Management** is an organization that runs the communications network. In a full scale implementation this is likely to be one or a consortium of the major communications service providers.

The VII **System Administrator** has responsibility for the management of the various assets and domains that need to be operated.

The **Network Users** are the various institutions and commercial entities that are using the System for their services. These include the government agencies gathering vehicle data, distributing traffic information and operating the transportation network, as well as the automakers maintaining contact with their customers.

The **External Data Sources** include a range of support services such as differential global positioning, security certificate authorities, and mapping services.

2.2 Related Standards

The VII System requires many standards to be adopted and enforced; the multitude of network users cannot operate without common formats. There will be communication standards within the system network, data standards within the vehicle, traffic data standards and dictionaries that are needed to unify the data across multiple jurisdictions. There will be interface standards for accessing the data, and the external sources will themselves impose standards that VII needs to adopt. This guide is concerned with the standards involved in the communications between mobile devices and from mobile devices to the infrastructure. Two standards are primarily involved:

- The IEEE 1609 standards (which defines the communications services)
- The SAE J2735 DSRC Message Set Dictionary (which defines the application level message that are exchanged)

The IEEE1609 standard breaks down into the following components:

- 1609.1, WAVE - Resource Manager
- 1609.2TM-2006, WAVE - Security Services for Applications and Management Messages
- 1609.3TM-2007, WAVE - Networking Services
- 1609.4TM-2006, WAVE - Multi-channel Operations

In addition, there is the IEEE 802.11p which is a draft amendment to the 802.11 standard to add Wireless Access in the Vehicular Environment (WAVE). It defines enhancements to 802.11 required to support ITS applications. This includes data exchange between high-speed vehicles and between the vehicles and the roadside infrastructure in the licensed ITS band of 5.9 GHz (5.85-5.925 GHz). IEEE 1609 is a higher layer standard on which IEEE 802.11p is based.[4]

802.11p will be used as the groundwork for DSRC, a USDOT project based on the European CALM system looking at vehicle-based communication networks, particularly for applications such as toll collection, vehicle safety services, and commerce transactions via cars. The ultimate vision is a nationwide network that enables communications between vehicles and roadside access points or other vehicles. This work builds on its predecessor ASTM E2213-03. The 802.11p task force predicts a published standard by late 2010.

The 1609 suite is concerned with communications between the various levels of the communication stack and the 802.11p standard is a modification of the standard WiFi 802.11 suite. The modification is principally to accommodate portable devices and low latency applications. Although the 1609 suite of standards is a crucial element of the DSRC data transmission, it is not the subject of this guide. If the reader wishes to be more informed on the nature of the wireless transmission, they are referred to the standard documentation (*www.standards.ieee.org*). The SAE J2735, DSRC Message Set Dictionary, is the subject standard for this guide. There have been multiple drafts developed over the years (2005 – 2009). The version that is used for this document is the second formally adopted edition of the standard, published 4 November 2009. This document describes the messages that are transmitted using the 1609 suite.

[4] IEEE 1609 - Family of Standards for Wireless Access in Vehicular Environments (WAVE). U.S. Department of Transportation (9 January 2006). Retrieved on 15 July 2007.

2.3 DSRC Design Philosophy

Design considerations are discussed in the following subsections.

2.3.1 Users

The users of the DSRC message set encompass a wide variety of skills and disciplines. There are the system designers that are working on in-vehicle and other PDA based applications. These developers have to design for a wide range of platforms, not just between manufacturers but between differing vehicle models that often change annually. These developers have the complex task of developing systems that can provide data in a timely manner and that also conform to these standards.

Communications between the vehicle and the infrastructure also imposes severe restrictions. First, the communication is sporadic at best. Under VII most of the road system will not have radio coverage; data can only be sent within the limited range of the roadside units (RSUs) or between nearby vehicles. This requires that the data within the vehicle be stored and transmitted when possible. The contrary case also applies in that data received by the vehicle must be stored and presented to the drivers at suitable locations and headings. Additionally, this location for uploading and downloading data is the same area where all the other vehicles are trying to do the same thing. Thus the transactions must be fast and accommodate multiple users with similar intentions.

The message set must also accommodate the needs and requirements of traffic engineers that are using the vehicle data as input to a variety of control mechanisms to assist them in their operations. These users of the message set view the information differently. They may be more familiar with the ATIS or the TMDD format where the information is normally delivered smoothed and filtered from ITS devices via a common format such as XML. This type of data (in particular, the traveler information messages found in ATIS) may be transmitted over DSRC media in an XML format using slower access methods than the time critical safety messages which this guide deals with. Other network users such as the automakers are likely to be communicating with their customers with encrypted files in their own formats.

2.3.2 Data

In addition to accommodating a wide range of users, data resolution is an issue. Although there are a large number of use cases for the data, it is not possible to cover all potential applications that have not yet been dreamed about. Indeed, nor is it possible to predict the additional data types that may occur in future automobiles; for this reason the data structures of the messages have been made extensible to accommodate new data types. Many users of traffic data have suggested that the data be smoothed or filtered as some applications do not need raw data. This discussion was expanded upon in the earlier days of VII development. The conclusion was that VII would adopt a policy of sending raw data to the network subscribers. Virtually every data element has an advocate for using data in the raw state and the subsequent design was based on this premise.

The design was also significantly influenced by the intermittent communication process inherent in short range transmission. In order to gather data from as much of the transportation network as possible, processes were developed for gathering information from the roadway as vehicles traverse the network, and concatenating these from transmission at the next suitable RSU. This imposes a burden on the message set to include traveler information for locations defined by geography and direction, and then send it to the vehicle for presentation to the driver at a later position and time.

The design of the message set is intended to use individual data elements across multiple applications. There are no data sets for individual applications. Rather, there is one set with multiple components that can be selected as needed by each application. For example, many applications need time, position and heading. The data set has developed these to meet all the

user requirements. The in-vehicle applications can read these data when they need it and accommodate the delivery format to their needs.

2.3.3 Privacy

Privacy has been embedded in the VII design philosophy since the beginning. There is an inherent conflict between the traffic engineers who would like all the data from all vehicles all the time, and the privacy advocates that need to be reassured that basic privacy principles are maintained. Some automakers stated that if they could not assure their customers that under no circumstances would they be tracked, then VII was not a viable project. This imposes requirements on the message set that need to be incorporated into the standard. Substantial negotiations took place to create a balance between the privacy needs and the requirements of the traffic engineering community. The results of these balances are reflected in the design of the messages and in some elements of the IEEE 1609 standards. For private vehicles, there are no data elements that can directly identify either a vehicle or a driver, nor can VII be used for tracking vehicles or for any enforcement processes.

The terms *private* and *public* as used here requires some definition. A public message, typically from a TMC, is sent to all vehicles. General ATIS event messages are broadcast in this way, encoded in XML, and sent using Internet Protocol (IP). Public messages are not sent to individual vehicles, they are all broadcast. Individual vehicles (individual OBUs) can selectively ignore or join streams of available data services and related users using services provided by the IEEE 1609 layer.

For safety messages exchanged between vehicles, broadcasting on the WAVE service channel using the WAVE short message (WSM), encoding format is used. Probe data from vehicles is considered public, as it can (once sent to an RSU) be accessed by multiple network users including public agencies; however this information is anonymous. Transactions between any institution or company and an individual user (such as a subscriber who has opted in) are considered private. Although all messages are authenticated, private messages are also encrypted.

2.3.4 Interfaces

The IEEE 1609 standards as they currently exist do not provide for a standardized Application Programming Interface (API) to the lower layer services they describe. Therefore, each embodiment will need to develop custom interfaces to these services, and they are likely to differ. This presents a problem for a guide such as this that hopes to illustrate ways to create proper messages and hand them off to and from the IEEE 1609 layers. To overcome this issue, the use of a common Berkeley socket style interface has been presumed when communicating with the IEEE 1609 layers. Code example have been built around this premise. This type of interface is widely known, understood, and easily built. It also was used in some of the field operation tests (FOTs), and it may come to pass in the future that 1609 formally adopts a similar solution.

As explained in further detail in the subsequent sections, the code examples have been implemented on a common PC using existing protocols. To emulate the broadcasting process of the IEEE 1609 WAVE WSM services, User Datagram Protocol (UDP) packets over IP are used to known socket/ports. This allows ready access to existing commercial off-the-shelf (COTS) protocol analysis tools to study and examine the messages described in this guide.

2.4 System Level Functional Services

The NSR v1.3.1 contains a complete listing of the system level requirements. The NSR defines a list of System Function Services. These are high level functions that support the function of the VII System and include the following.

2.4.1 Advisory Message Distribution

The VII Advisory Message Distribution Service (AMDS) provides Network Users with the ability to send advisory messages to Mobile Users in targeted geographic locations for a specified time interval. The AMDS provides the Roadside Infrastructure User the ability to send advisory messages to Mobile Users in the User's local geographic area, and provides the Public Service Mobile User (PSMU) the ability to send advisory messages to Roadside Infrastructure in the PSMU's local area. Messages are delivered in a prioritized order. It also supports cancellation of advisory messages by an authorized or originating Network User or Roadside Infrastructure User.

Thus an application that considers a traveler information message will typically originate in an agency's TMC. The application will likely need to restrict its distribution to those specific RSUs that are owned or operated by that agency. For example, a city's application will not likely extend onto a state operated interstate. Although not yet defined, the access rights at the RSU level will likely be restricted by the owner.

DSRC Safety messages which are distributed in this way include the Roadside Alert which can be sent from either an RSU (typically in the case of standing construction warnings or wide area incident events from the network) or from a public safety OBU (typically a responder moving in the local area or an on-scene responder sending situational warning to arriving travelers).

2.4.2 Communications

The VII System will provide communications facilities sufficient to allow Mobile Users to exchange information with the Network User, with the Roadside Infrastructure User, and with other Mobile Users. These facilities may provide varying levels of service, depending on the operational characteristics of the Mobile User, its environment and the type of information being exchanged.

The focus of the DSRC Safety messages is mostly *vehicle to vehicle* (V2V) in nature, although *vehicle to roadside* (V2R) and *roadside to vehicle* (R2V) exchanges also exist. The DSRC Safety messages illustrated in this guide contain all three styles. The `BasicSafetyMessage` is a V2V type, while the `ProbeVehicleData` report message is a V2R message. The `RoadSideAlert` message can be either V2V or R2V.

All messages that are considered here are broadcast and thus the message designer needs to be aware of the consequences. For example, unlike traveler information that is sent to a variable message sign (VMS), these messages need to include information on the location and direction of display. There are likely to be some generally applicable messages, but most require the information on the effective location and direction that is relevant for the data being transmitted.

2.4.3 Information Lookup

The VII System's Information Lookup Service provides facilities for Network Users, Administrative Users, and selected VII subsystems to determine capabilities, locations and connectivity information of selected (non-mobile) VII components, sufficient to enable them to communicate with and control those VII elements. The message developer will need to use this service to gather the RSU addresses that are relevant to the message content.

This is outside the scope of the DSRC Safety messages and their applications and is not illustrated by this guide.

2.4.4 Management

The VII Management Service provides facilities to maintain and monitor the performance, security and configuration of the non-mobile components of the VII System. This includes:

- Tracking and management of the VII System configuration
- Provisioning and configuration of non-mobile VII System components

- Detection, isolation and correction of selected non-mobile VII System component and service problems
- Monitoring of VII System and subsystem performance.

This is outside the scope of the DSRC Safety messages and their applications and is not illustrated by this guide.

2.4.5 Map Element Distribution

The VII Map Element Distribution Service (MEDS) distributes approved localized micro maps (small maps containing accurate roadway geometry and lane features) to Mobile Users in support of lane-based safety applications. The MEDS uses a set of reference maps, augmented with map elements derived from or verified by probe data, to generate and update these micro maps. The approved micro maps are distributed to Mobile Users near the map location as needed to support end-user applications.

This is part of the scope of the DSRC Safety messages, and supported by existing messages for map and roadway geometries (in conjunction with signal timing and state messages). However, this guide does not illustrate any use of these messages at this time.

2.4.6 Positioning

The VII System Positioning Service acquires, maintains, and distributes positioning and position correction information to VII subsystems that require such information, and to mobile users that may make use of such information. The Positioning Service will use a variety of available external positioning services as well as mechanisms such as inertial navigation to provide positioning data (informative).

This is part of the scope of the DSRC Safety messages. The DSRC Safety messages currently provide various message formats to support differential corrections for Global Positioning System (GPS) and other precision positioning systems. However, this guide does not illustrate any use of the corrections messages at this time.

2.4.7 Probe Data

The VII Probe Data Service collects anonymous probe data from all Mobile Users, and distributes it to any authorized Network User or Roadside Infrastructure User that requests it. Probe messages will conform to the SAE J2735 standard (see the *MSG_ProbeDataManagement* and the *MSG_ProbeVehicleData* messages and the appropriate application annex). Probe data distribution will occur in near real time; the VII System keeps probe data only long enough to distribute it to requesting authorized users. Data that is not requested is lost.

This is part of the scope of the DSRC Safety messages, and the *MSG_ProbeVehicleData* message is illustrated in greater detail in section 5. This guide does not illustrate any use of the probe management message at this time.

2.4.8 Roadside Infrastructure Support

The VII Roadside Infrastructure Support Service provides data collection, processing and formatting services for a Roadside Infrastructure User (Local Data Provider) which is unable to provide properly formatted messages.

This is outside the scope of the DSRC Safety messages and their applications and is not illustrated by this guide.

2.4.9 Security

The VII Security Service provides for *defense in depth* protection of its hardware, software and information components against hackers, unauthorized users, and physical or electronic sabotage. The VII Security Service also maintains network administration control of authorized user privileges and protects the confidentiality of user information. It will detect, isolate and correct violations of VII System security. For further background regarding the conceptual view of how VII Security objectives will be implemented, refer to the VII Systems Security Plan.

This is outside the scope of the DSRC Safety messages and their applications and is not illustrated by this guide.

2.4.10 System Time

The VII System Time Service acquires and maintains timing information for Mobile Users and those VII Subsystems that require it. The System Time service will likely use GPS time as its foundation (informative).

This is outside the scope of the DSRC Safety messages and their applications and is not illustrated by this guide. However, the message sets themselves are aware of, and make use of, this requirement. Many of the message exchanges assume that the receiving unit knows the current time to a precision of well under a second, and the current minute is typically known and not sent in the data.

2.5 VII Subsystems

The reader of this guide should be aware that these functions exist and their detailed requirements are in the NSR. Of specific concern is of course the communications. The NSR defines six major subsystems:

- Certificate Authority
- Enterprise Network Operations Center
- Private Mobile User Segment (PMUS)
- Public Service Mobile User Segment (PSMUS)
- Roadside Equipment
- Service Delivery Node

Messages that traverse the VII System may be digitally signed or encrypted using public and private keys issued by a Certificate Authority (CA). The CA issues keys and corresponding certificates and publishes lists of revoked certificates so that all entities in and users of the VII System can verify that a particular message is from a valid source and can be trusted.[5]

The Enterprise Network Operations Center, Service Delivery Node and Roadside Equipment together make up the VII infrastructure. The Private Mobile User Segment defines the communication component.

The Private Mobile User Segment (PMUS) is a set of processing, communications, and interface functions that permit a Private Mobile User to interact with the VII System's messaging, security, management and communications functions. The PMUS provides an interface to the Private Mobile User and a (WAVE/DSRC) radio interface to other elements of the VII System (RSUs, other PMUS, and PSMUS). The PMUS functionality is contained in a mobile system (vehicle, or non-original equipment such as an aftermarket, portable, handheld, etc.) but may be implemented in many different ways that are not specified (informative).[6]

[5] National System Requirements v 1.3.1 April 2008

[6] National System Requirements

The VII Roadside Equipment (RSE) is the equipment positioned along highways, at traffic intersections, and in other locations to support wireless communications between VII-compliant Mobile Users and the VII Infrastructure. Each VII RSE Subsystem includes a WAVE (DSRC) radio, a positioning system, processor, and limited storage; an interface to Roadside Infrastructure Users; and a network interface to connect with the VII Infrastructure. The RSE is actively connected to a primary SDN Subsystem and connects to an alternate SDN Subsystem if its primary SDN Subsystem is unreachable.

The Service Delivery Node (SDN) Subsystem contains server platforms, data stores, and software systems that support VII System data distribution and communications services. Each SDN Subsystem includes a Network Access Point (NAP) to support connectivity to: ENOC Subsystems, multiple nearby RSE Subsystems, multiple nearby Network Users, and other SDN Subsystems. Collectively, the SDN NAPs and ENOC NAPs and communications infrastructure compose the VII Infrastructure backbone.

2.6 VII External Interfaces

The VII system acts as a conduit from a range of external users that need to communicate with the vehicle. Examples of these users include such entities as the automakers that might contact the vehicle or its owner, the VII network administrator that may wish to modify systems elements, and public agencies providing traffic data. The NSR refers to applications in the vehicle as the Mobile Transaction Agents and defines these 13 communication requirements as follows:

1) The VII System shall provide means for communicating an unaltered message without guaranteed delivery from a Mobile Transaction Agent to a Network Transaction Agent.
2) The VII System shall provide means for communicating an unaltered message without guaranteed delivery from a Mobile Transaction Agent to a Local Transaction Agent.
3) The VII System shall provide means for communicating an unaltered message without guaranteed delivery from a Network Transaction Agent to a Mobile Transaction Agent which has provided its network address to the Network Transaction Agent.
4) The VII System shall provide means for communicating an unaltered message without guaranteed delivery from a Local Transaction Agent to a Mobile Transaction Agent which has provided its network address to the Local Transaction Agent.
5) The VII System shall provide means for communicating an unaltered message without guaranteed delivery from a Local Transaction Agent to a Network Transaction Agent.
6) The VII System shall maintain a communications session between a Mobile Transaction Agent and Network Transaction Agent until either the Network Transaction Agent or the Mobile Transaction Agent terminates the active session, or a configurable time elapses.
7) The VII System shall maintain a communications session between a Mobile Transaction Agent and Local Transaction Agent until either the Local Transaction Agent or the Mobile Transaction Agent terminates the active session, or a configurable time elapses.
8) The VII System shall provide a means for determining the network address of a Network Transaction Agent or Local Transaction Agent based on an associated logical identifier.
9) The VII System shall provide a means for defining and maintaining logical identifiers for the Network Transaction Agent and Local Transaction Agent.
10) The VII System shall provide a means for an authorized Transaction Agent (Local Transaction Agent, Network Transaction Agent) to provision its logical identifier.
11) The VII System Communications Service shall support a minimum of four classes of service that provide four levels of quality of service.
12) The VII System shall provide for means for communicating a message without guaranteed delivery from one Mobile Transaction Agent to another Mobile Transaction Agent.
13) The VII System shall notify the Mobile Transaction Agent of when it is able to provide communications services.

Currently there are no timing, latency or performance requirements specified. This leaves the developer without guidance on these requirements. This is an important omission in a near real time system that mixes fast time critical events such as the GPS timing signal to listen to the control channel together with slower events such as TMCs sending out traveler information once per minute. Until definitive requirements are developed, the developer is encouraged to design systems where the mobile user to roadside link is developed in as near real time as is feasible.

2.7 Interaction with Other Wireless Protocols

5.9 GHz DSRC is a secure wireless interface used to support VII applications. It provides a high speed, short range wireless interface between vehicles and surface transportation infrastructure. It enables the rapid communication of vehicle data and other content between OBUs (or mobile users) and RSUs. It supports both inter-vehicle and vehicle to infrastructure communications, and operates in a licensed frequency band allocated by the Federal Communications Commission (FCC) and is based upon the WiFi standard IEEE 802.11. Specifically, 802.11p defines the DSRC Lower Layers and is known as Wireless Access in Vehicular Environments (WAVE).

The general process for communications in this environment is taken from material from TechnoCom.[7] The following steps are required for successful communication.

Registration

- Applications must be registered in both the RSU and OBU
- Registration parameters include Application ID (AID) and other application information

Control Channel Monitoring

- OBUs monitor the control channel to listen for application announcements and broadcast safety messages
- The control channel must be monitored every 100 mSec for a minimum required amount of time (the minimum time may be variable based on load)

Application Announcements

- Applications may be announced periodically or on demand within a provider service table (PST)
- PST contains AID among other application parameters
- Periodic announcements are used on applications such as toll collection
- On demand announcements are used to invite DSRC devices to participate in an application based on events (e.g., collision avoidance)

Application Initialization and Execution

- Applications are initialized by matching the locally registered AID with an advertised AID (application announcement) received on the radio link
- Application data exchange is usually executed on a service channel
- 6 service channels are available

The above initialization steps represents the more complex case when a device needs to join a group of other devices operating in a shared application framework. A V2V safety application example of this would be a platoon of cooperating vehicles. A simpler set of protocols also exists in the WSM sent over the control channel (and is discussed further below). In the above this is referred to as *monitoring the control channel*. Messages in this format are typically

[7] TechnoCom VII public meeting presentation on DSRC 27 July 2005 – Justin McNew

unacknowledged[8] and broadcast to the surrounding vehicles or devices (OBUs or RSUs) within range. A V2V safety application example of this would be a basic safety message which is periodically broadcast from moving vehicles.

As noted, there are two types of wireless DSRC data communication:

- **WSM – Wave Short Message:** The WSM is useful for broadcast applications and applications that do not require infrastructure or extensive routing and addressing mechanisms. It does not require continuous access to the VII network infrastructure and thus is particularly suited for short safety messages between vehicle and roadside devices. WSM primarily supports the near instant exchange of safety information, generally localized to a collection of two or more vehicles or between a specific RSU and nearby vehicles. Example applications include intersection collision avoidance, traffic signal warning, and emergency electronic brake lights. The three messages used in this guide are all sent using the WSM, with a payload encoding of ASN DER. The WSM provides no concept of a link layer at this time, therefore all messages must fit within the size limits of a single packet.
- **Internet Protocol version 6 (IPv6) Datagrams:** A datagram is a single packet using IP and is the most common method for routing packets through a network such as VII. It facilitates VII network build-out using commercial off-the-shelf equipment. It also supports the exchange of safety and other content provider data over a network that can be readily established based on existing communications infrastructure. It uses the industry standard link layer to reliably send messages which are larger than the packet size. Example applications include traveler information, parking payment, toll payment, fleet vehicle monitoring, and remote vehicle diagnostics. Other messages defined in the DSRC message set are sent using this protocol, typical with XML payload encoding.

The DSRC structure has one control channel and multiple service channels. Every 100 mSec,[9] all OBUs listen to the service channel for broadcasts. The table[10] below indicates a preliminary channel allocation for the various message types. Each of the service channels is intended for use for differing power levels. For example, one channel may be allocated a higher power level and be used by fire trucks to request signal priority, whereas another channel may be used for payment and need low power to prevent adjacent lane reads. However, the precise use of each of the channels has yet to be determined.

Table 1 – DSRC Message Protocol Types Mapped to Service Channels

Message Type	Control Channel	Service Channels (in motion)	Service Channels (stopped)
Service Announcement	✓		
Broadcast WSM (with BER-DER payloads)	✓	✓	✓
Unicast WSM (with BER-DER payloads)	permitted but rare	✓	✓
UDP/IPv6 (with XML payloads)		✓	✓
TCP/IPv6 (with XML payloads)		permitted but rare	✓

[8] Any acknowledgement occurs above the provided network layers in the application layer, similar to the way UDP packets can be used in more complex exchanges.

[9] This is highly dependent on the local implementation design requirements.

[10] Doug Kavner – Presentation VII Coordination Meeting 4 October 2005, edited to add payload details.

The IEEE 1609 suite of standards provides a range of services that are required for DSRC operation. Figure[11] 2 illustrates the mapping between the OSI model and the various 1609 standards that contribute to the WAVE model. The principle functions of the 1609 components are as follows:[12]

- IEEE 1609.1 describes the services and interfaces, including security and privacy protection mechanisms, associated with the DSRC Resource Manager operating at 5.9 GHz band authorized by the FCC and to satisfy the ITS wireless communications requirements.
- IEEE 1609.2 defines secure message formats and processing of secure messages, within the DSRC/WAVE system:
  - defines methods for securing WAVE management messages and application messages, with the exception of anonymity-preserving vehicle safety messages
  - describes administrative functions necessary to support the core security function
- IEEE 1609.3 defines services, operating at the network and transport layers, in support of wireless connectivity among vehicle-based devices and between fixed roadside devices and vehicle-based devices using the 5.9 GHz DSRC/WAVE mode.
- IEEE 1609.4 describes multi-channel wireless radio operations that uses the IEEE 802.11p, WAVE mode, medium access control and physical layers, including the operation of control channel and service channel interval timers, parameters for priority access, channel switching and routing, management services, and primitives designed for multi-channel operations.
- IEEE 802.2 is the standard defining *Logical Link Control* (LLC), which is the upper portion of the data link layer for *local area networks*. The LLC sublayer presents a uniform interface to the user of the data link service, usually the *network layer*.
- IEEE 802.11 is the Standard for Information Technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.[13]
- IEEE 802.11p is a modified version of part 11 designed to accommodate WAVE functionality.

[11] TechnoCom Presentation – John Moring Status and Plans May 2008, edited to correct.

[12] Tom Kurihara IEEE DSRC Application Services Presentation July 2007.

[13] *www.standards.ieee.org*

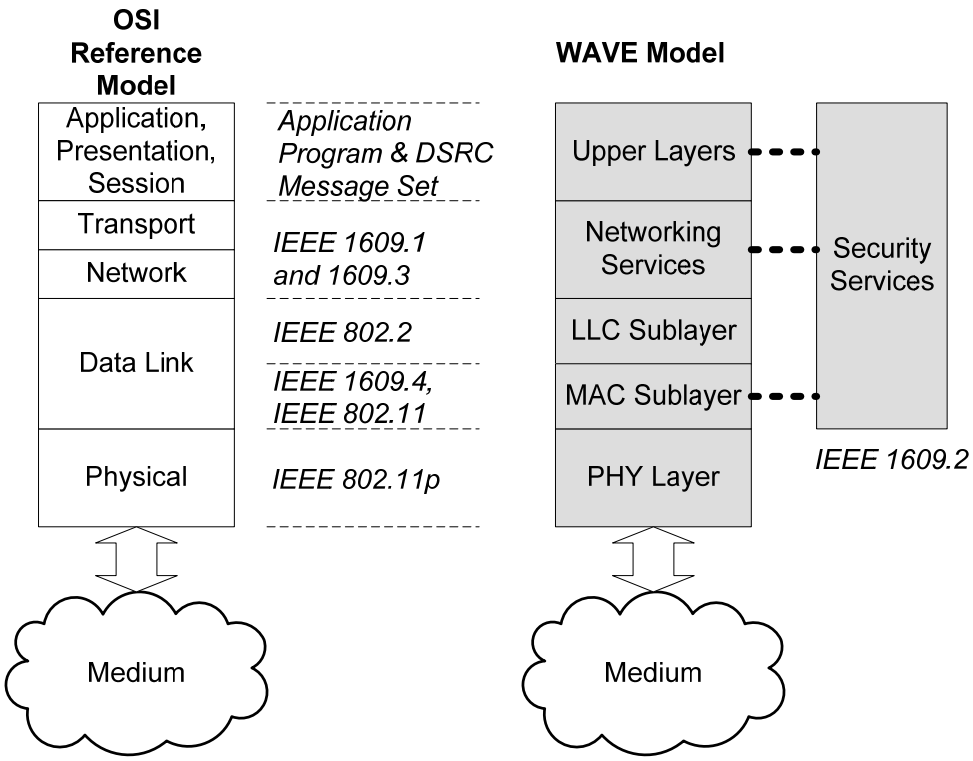

Figure 2 – OSI vs. WAVE Protocol Stack

Security is a principle service of IEEE 1609. The DSRC service uses Public Key Infrastructure (PKI) to sign and/or encrypt DSRC data. This requires both a central certificate authority and a collection of certificates that are kept in the OBU and RSU. A collection of certificates is used to prevent vehicle tracking by use of the certificates. PKI requires the broadcast of a Certificate Revocation List (CRL). Another aspect of the privacy and non-tracking of vehicles becomes apparent here as the MAC address needs to be randomly changed to prevent vehicle tracking. In cases where the data messages do not need to be encrypted, they are still authenticated with digital signatures to protect against security threats. Although the private messages between, for example, the automakers and their customers, or between a driver and opt-in traffic data and navigation services, are likely to be encrypted by methods proprietary to the provider.

Messages have two priority parameters, importance and urgency, defined as follows:

> **Priority** is a measure of the societal impact per the edict of the FCC. The greater the potential for saving lives the higher the priority.
>
> **Urgency** is a measure of the requirement for a low communication latency; if a message requires a quick transfer from sender to listener, then it has a higher urgency.

Message priority is still under consideration; however, the current proposals are defined in Table 2 below.[14] It should be emphasized that these are proposed values and should only be used for guidance.

These are the principle interfaces and some of their interactions with which the DSRC message set must consider. The primary wireless interface to the lower layers is IEEE 1609, which in turn exposes other layers. The IEEE standard establishes certain services to be provided, but it does not at this time define a standardized programming interface (an API) to expose those services. This issue is treated further in the next section.

[14] Ron Hochnadel Status Report to the IEEE 1609 Working Group 29 April 2008

Table 2 – Proposed Message Priorities

Importance Level from FCC Policy Description	Description (When to apply a specific urgency level)	Example(s)	Reception Latency (Urgency)	Message Priority	Access Category (IEEE 802.11 Std)	Channel (Suggested)
1 = Safety of Life Those Messages and Message Sets requiring *immediate* or *urgent* transmission	Emergency impact mitigation and injury avoidance/mitigation	Crash-Pending Notification	< 10 mSec	7	3	Control
	Emergency potential-event impact and/or injury mitigation and avoidance	Pre-Crash	< 10 mSec	7	3	Control
	Urgent Warning Events (using Event Flags)	Hard-Brake (Collision Warning, EEBL, Anti - Lock, etc.) and Control Loss	< 10 mSec	7	3	Control
	Urgent warning of impending local situation	Emergency Vehicle Approaching	10 to 20 mSec	5	2	Control
	Situation-based status information of uninvolved local interest	Situation Ahead (e.g., Emergency Vehicle, Accident)	10 to 20 mSec	5	2	Service
	Situation-based status information of uninvolved local interest	Situation Ahead (e.g., Emergency Vehicle, Accident)	10 to 20 mSec	5	2	Service
	Potential-situation information of uninvolved local interest	Probable-situation (e.g., Rapidly deteriorating dangerous conditions)	10 to 20 mSec	5	2	Service
	Possible-situation information of uninvolved high-latency local interest	Possible-situation (e.g., Deteriorating dangerous conditions)	> 20 mSec	3	1	Service
2 = Public Safety (Safety not in 1) Roadside Units (RSUs) and On-Board Units (OBUs) operated by state or local governmental entities that are presumptively engaged in public safety priority communications (Includes Mobility and Traffic Management Features)	Urgent public safety downloads (Intersection Information)	SPAT (Signal Phase and Timing)	< 10 mSec	6	3	Control
	Urgent public safety data transactions, exchanges	Electronic Toll Collection	< 10 mSec	5	2	Service
	Periodic public safety status information	Heartbeat message	< 10 mSec	4	2	Control
	Public safety geospatial context information	GID message (Geospatial Context)	< 10 mSec	4	2	Control
	Semi-urgent public safety link establishment	Lane Coordination; Cooperative ACC	< 10 mSec	4	2	Control
	Public safety GPS correction information	GPSC message (GPS Correction)	< 10 mSec	4	2	Control
	Periodic less frequent public safety status information	Heartbeat (at lower frequency)	< 10 mSec	3	1	Control
	Semi-urgent public safety system enabler	Localized Geometric Intersection Definition Download	10 to 20 mSec	3 (0)	1	Service
	Semi-urgent public safety data and application enabler	Services Table, Digital Map Download	> 20 mSec	3 (0)	1	Service
	Important Traffic Management status information enabler	Highway Closed Ahead	> 20 mSec	3 (0)	1	Service
	Important Announcement of Services	WSA message (Wave Service Announcement)	> 20 mSec	3 (0)	1	Service
	Semi-important Traffic Management enabler	General Traffic Information Download	> 20 mSec	3 (0)	1	Service
	Non-urgent Traffic Management Foundational Data	Probe Messages, Localized warning zones update	> 20 mSec	3 (0)	1	Service
3 = Non-Priority Communications (not in 1 or 2) Fleet Management of Traveler Information Services and Convenience or Private Systems	Urgent, private mobility message	Off-Board Navigation Reroute Instructions	< 10 mSec	2	0	Service
	Urgent, private and commercial electronic transactions	Electronic Payments	< 10 mSec	2	0	Service
	Semi-urgent, private mobility data exchange	Private mobility applications (e.g., GPS based driving instructions)	10 to 20 mSec	1	0	Service
	Semi-urgent, private and commercial electronic transactions	Commercial and private e-commerce applications	10 to 20 mSec	1	0	Service
	Important, private and commercial electronic transactions	Large commercial transactions (E-Commerce)	10 to 20 mSec	1	0	Service
	Important, private mobility data exchange	Commercial and private offers, directions	10 to 20 mSec	1	0	Service
	Background, private mobility data exchange	Area map download or upgrade	> 20 mSec	1	0	Service
	Background, private data downloads and upgrades	Database download	> 20 mSec	1	0	Service

2.8 Implementation Considerations

Implementation of the DSRC message set into an environment which supports the lower layers of IEEE 1609 and other standards is at this time a unique and custom design effort for each device maker. This is due not only to the nature of the product (typically an embedded real time device implementing specific applications) but made more complex by the lack of existing APIs for the IEEE 1609 layer. In this guide that deals with only some of the message payloads carried by the IEEE 1609 layer, and not that layer itself, some design choices regarding implementing the IEEE 1609 layer have had to be made. This subsection outlines some of these considerations.

All three of the representative messages discussed in this guide are transmitted over the service channel using the WSM protocol. The message payload when transmitted in this fashion is always encoded in the ASN.1 form referred to as BER-DER, with its data content as defined by the DSRC message set standard. It should be noted that some XML encoded content is also defined by this standard which is sent over datagrams. Other standards are presumed to exist over time which define additional messages that will co-exist with the DSRC safety messages.

The challenge in this guide is to clearly demonstrate how to correctly construct and decode these messages, and to send them over a standard PC environment in some way. Readily available PCs do not have a network card that supports the WSM; however, they typically do have an Ethernet card of some type. Therefore, we have elected to use a simple UDP packet to represent the broadcast nature of the WSM packet on a single service channel. Within the UDP packet, the data payload to be sent will exactly match that defined by the SAE DSRC message set and its ASN.1 encodings. These packets can be sent in a broadcast mode to the PCs' local subnet so that nearby machines can recover them with the same general format as defined in the WSM process. When IP datagrams need to be sent (at this point this is an *if*, as none is yet needed) then the data payload to be sent will exactly match that defined by the SAE DSRC message set and its XML encodings.

Because the IEEE 1609 standard does not at this time have an API, we have chosen to use the common Berkeley sockets style as a default API for the code examples as the interface means to the lower levels. This provides a readily understood interface familiar to a wide developer audience, and allows them to sidestep the unresolved issues of what (or how) supporting services that can reasonably expect the lower layers to provide. It also improves the likelihood the example code base can be re-used on non-PC platforms. (The Berkeley socket is by far the most common means to implement similar protocols and widely available on many systems.) As needed, additional management layer data that the IEEE 1609 protocol may need is noted in the examples. Any number of good reference texts regarding the Berkeley socket system, as well as explaining the details of TCP/IP connections can be found online.

As a further observation, by using this choice to implement the WSM layer in common UDP and TCP packets, a large number of COTS tools can be used as protocol monitors to recover and decode individual packet as they are sent. In the development of the code examples, commonly used tools are used to grab packet traffic and display it for further review and analysis.

Note that at this time no attempt has been made to model issues of channel selection, suitable service channel dwell times, or of the loss of message traffic due to a receiving device being on the wrong channel. These issues are of great importance to the overall system success, but are outside the scope of this document which deals with the message encoding content. However, simple events such as message loss or message collision are still present and are modeled in the Berkeley socket system in a similar fashion as in the WSM system.[15] Other traffic on the user's PC will still serve to create such effects.

[15] In both systems, corrupted packets are simply not delivered to the next (upper) layer; so the listener device will simply not see any such bad packets. In the DSRC WSM, there is also a cryptographic decoding function which may cause incoming packets to be rejected; the effect is the same, no packet is handed over to the application layer.

The WSM protocol defines a 3 bit (7 level) priority level to be assigned to each message. Some DSRC messages set this in varying ways (in some of the messages this is set based on the message content). The UDP system also defines a 3 bit priority system. We will use the UDP system to model the WSM system.

The WSM protocol uses a MAC value for unique device identification and for limited addressing, however this value randomly changes to preserve vehicle anonymity. Ethernet, at least that found on the typical PC, uses a permanent MAC value assigned by the manufacturer for each NIC device, and it is used in a similar fashion. For illustrating the DSRC message set, we will use the PC machine's address and not change it (the changing in the WSM occurs in a lower layer and the applications need not be aware of it, so this decision does not affect validity of the created messages). For identification values found within the message set, notably the BSM vehicle ID, we will use the lower 4 bytes of the IP address of the machine or a user selected value.

Because of the repetitive un-acknowledged broadcast nature of the WSM messages, some method is needed to detect that a specific DSRC safety message has been received in each OBU previously (the duplicate messages problem). Ideally, a lower layer would provide a Cyclic Redundancy Code (CRC) value for the data payload of each message; however, without any form of standardized API this service may not be present. As a means to accommodate this need, a CRC data element covering the message payload is present in some messages.[16] This value can be compared with previously received CRC values to detect messages with duplicate content. Note that this value does not change regardless of PKI or other cryptographic changes that do affect the lower layers and message framing.

[16] The precise format used and the encoding algorithm is defined in the standard itself and examples are provided in later sections.

3. Application 1
Basic Safety Message – Intelligent Brake Light Warning

3.1 Description

The Basic Safety Message (BSM) is used in multiple safety applications in each vehicle. These applications are largely independent of each other, but all make use of the incoming stream of BSMs from surrounding (nearby) vehicles to detect potential events and dangers. One of these applications, referred to as *intelligent braking*, attempts to compare the motion vector and vehicle status found in the BSMs received from other vehicles with its own, in order to detect potential rear-end collisions and either warn the driver or take corrective action[17] such as enhancing brake actions and pre-tensioning seat belts. This application is used as the example in this section to explain and to demonstrate encoding and decoding the BSM content.

In this application the fundamental need is to receive the BSM from other nearby vehicles, and then detect and warn the driver if any of the other vehicles seem to be decelerating or moving in a general direction that would indicate a need for the receiving vehicle to brake. Detection algorithms are expected to vary considerably among developers (and are not covered here), but at a gross level, detection of acceleration differences of ~0.3 g or more combined with a similar motion vector, the other vehicle's brake lights being active, while being ahead in the same lane, provides a rudimentary classification system. Upon detection (which again is outside of scope here) various actions may occur, ranging from passive driver alerting to actively changing vehicle dynamics and parameters including disabling any cruise control and active braking.

In the basic system defined by the SAE J2735 standard, each moving vehicle updates and sends its own BSM every 100 mSec over the WSM channel.[18] Other nearby devices (typically vehicles but potentially also RSUs) detect this broadcast and process it as they see fit (running whatever safety applications they wish). There is no handshaking or acknowledgment between the devices; there is no *association* or *join* process (tracking the collection of *in-view* vehicles is the responsibility of each receiver). A temporary value in the message body allows correlating BSMs to a specific vehicle for short periods of time (needed for trajectory estimates). The BSM message is expected to be the most statistically frequent message seen over the airwaves.

The BSM itself consists of two general sections, defined as Part I and Part II. The information in Part I is always sent, using a combination of DER encoding and some octet binary large object (blob) encoding. Part I contains information regarding position, motion, time, and general status of the vehicle. Part II is sent when needed and with content which may vary. The definition allows many additional (and therefore optional) data items to be included in the Part II content when the sender feels these are useful. This information is intended to assist the receiving devices in further processing. For example, an event trigger word may be sent, indicating that the sending device has determined an event has occurred (such as hard breaking or impending violations of various types). The specific details of what can and cannot be sent (as well as its encoding) can be found in the standard.

[17] It should be kept in mind that the purpose of the selected applications in this guide is simply to show the details of the message set, not to create a life-like application with all the attendant logic, real time constraints, and human factors that a real application would entail. Hence, actual application level logic (in this case comparing the motion of multiple vehicles in a short period of time) is kept to a minimum.

[18] While there have been proposals for adaptive sending rate algorithms to better preserve bandwidth (especially during periods of slow moving dense traffic), at this time none is adopted in the standard and a static transmission rate of every 100 mSec is used.

3.2 Relevance to the VII Architecture

With regard to the national architecture, the Basic Safety Message is the critical message that provides data for a variety of use cases. The BSM provides situational awareness between vehicles; this means that each vehicle is aware of the position, speed, and heading of all vehicles within range. In addition to the Brake Light Warning application used in this guide, some example use cases that make use of this facility include:

- **Blind Spot Warning** - This application warns the driver when he intends to make a lane change and his blind spot is occupied by another vehicle. The application receives periodic updates of the position, heading, and speed of surrounding vehicles via vehicle to vehicle (V2V) communication. When the driver signals a lane change intention, the application determines the presence or absence of other vehicles in his blind spot. If the presence of a vehicle in his blind spot is detected by the application, a warning is provided to the driver. A variation on this is the merge assist where the same processes are used at merge points.
- **Cooperative Adaptive Cruise Control** (ACC) - Cooperative adaptive cruise control uses V2V communication to obtain lead vehicle dynamics and enhance the performance of current adaptive cruise control. Enhancements that could be made to ACC include stopped vehicle detection, cut-in vehicle detection, shorter headway distance following, and improved safety. The application can be enhanced by communication from the infrastructure, which could include intelligent speed adaptation through school zones, work zones, off-ramps, etc.
- **Cooperative Collision Warning** - Cooperative collision warning collects surrounding vehicle locations and dynamics and warns the driver when a collision is likely. The vehicle receives data regarding the position, velocity, heading, yaw rate, and acceleration of other vehicles in the vicinity. Using this information along with its own position, dynamics, and roadway information (GID data), the vehicle determines whether a collision with any vehicle is likely. In addition, the vehicle transmits position, velocity, acceleration, heading, and yaw rate to other vehicles.
- **Cooperative Forward Collision Warning** - Cooperative forward collision warning systems are designed to aid the driver in avoiding or mitigating collisions with the rear end of vehicles in the forward path of travel through driver notification or warning of the impending collision. The system does not attempt to control the host vehicle to avoid an impending collision.
- **Emergency Vehicle at Scene Warning** - Emergency vehicle transmits a signal to warn oncoming motorists from either direction that there are emergency vehicles ahead.
- **Lane Change Warning** - This application provides a warning to the driver if an intended lane change may cause a collision with a nearby vehicle. The application receives periodic updates of the position, heading, and speed of surrounding vehicles via V2V communication. When the driver signals a lane change intention, the application uses this communication to predict whether or not there is an adequate gap for a safe lane change based on the position of vehicles in the adjacent lane. If the gap between vehicles in the adjacent lane is not sufficient, the application determines that a safe lane change is not possible and provides a warning to the driver.
- **Pre-Crash Sensing** - Pre-crash sensing can be used to prepare for imminent, unavoidable collisions. This application could use VII communication in combination with other sensors to mitigate the severity of a crash. Countermeasures may include pre-tightening of seat belts, airbag pre-arming, front bumper extension, etc.

These example applications all depend on V2V communication and have significant potential for saving lives. The basic information that is needed for these applications is common to all. Situational awareness (position heading and speed of the other vehicles) is a common factor that allows this message to be sent once and utilized by multiple applications.

3.3 Communication Parameters

Intelligent Brake Light Warning

Minimum Required Communication Range	From nearby (adjacent bumper to bumper vehicle traffic) to approximately 100 meters distance
Message Direction	One way broadcasts coming from all operating and moving vehicles
Broadcast Interval	Updated messages reflecting a vehicle's current values to be sent at 10 per second nominal (once every 100 mSec)
Priority Assignment Criteria	Sent over WSM using nominal priority values Unresolved: Issues at this time to increase transmission priority if certain events have occurred (such as a self determination of a rapid deceleration event)

3.4 How to Use the Message

In this application example we create a valid but simple BSM message using the two required top level data elements defined in the message. We then encode this into ASN in both a memory structure and then into a valid message, using the ASN library to accomplish this. We also briefly look at the support tools used to create the message and to view the resulting encoding and become familiar with their use. In the next applications, these details are reviewed in greater depth (particularly with regard to creating ASN encodings).

The ASN.1 definition of the BSM (taken from the SAE J2735 standard) is provided below. We now examine how this is converted by the chosen ASN.1 tool to produce the ASN library which the application can use.

```
BasicSafetyMessage ::= SEQUENCE {
   -- Part I
   msgID       DSRCmsgID,               -- 1 byte

   -- Sent as a single octet blob
   blob1       BSMblob,

   -- Part II, sent as required
   -- Part II,
   safetyExt   VehicleSafetyExtension   OPTIONAL,
   status      VehicleStatus            OPTIONAL,

   ... -- # LOCAL_CONTENT
   }
```

The message consists of only four top level discrete parts, all also defined by the standard. This is not particularly informative, given that all the main content is in the item defined as an octet blob (the element `BSMblob`) or found in Part II, and that you must seek out the two elements in Part II to discover what is in Part II.

> **Hint**: Moving about in the SAE J2735 standard is easily done in any electronic copy of the document by following (usually via **Ctrl-click** or **Shift-click**) the blue links to jump to the different definitional areas. It may be useful to keep a copy of the standard up in another window when reviewing these sections.

We will now spend several pages examining the subtleties of this message and its components. In the next subsection, we delve into how this is represented in the C language when supported by the ASN library routines.

For the purpose of clarity, below is the *single data element* verbose view of the same ASN (used when the developer specifically wants to encode each data item independently in ASN DER; it is not used in the BSM when transmitted over WSM). The data content and order is the same, however here the bulk of the Part I content is not constructed as a single octet blob (a topic discussed further in a moment). As a result, this variant is substantially longer and larger.

```
BasicSafetyMessageVerbose ::= SEQUENCE {
   -- Part I, sent at all times
   msgID       DSRCmsgID,            -- App ID value, 1 byte

   msgCnt      MsgCount,             -- 1 byte
   id          TemporaryID,          -- 4 bytes
   secMark     DSecond,              -- 2 bytes
   -- pos      PositionLocal3D,
   lat         Latitude,             -- 4 bytes
   long        Longitude,            -- 4 bytes
   elev        Elevation,            -- 2 bytes
   accuracy    PositionalAccuracy,   -- 4 bytes

   -- motion   Motion,
   speed       TransmissionAndSpeed, -- 2 bytes
   heading     Heading,              -- 2 bytes
   angle       SteeringWheelAngle,   -- 1 bytes
   accelSet    AccelerationSet4Way,  -- 7 bytes

   -- control  Control,
   brakes      BrakeSystemStatus,    -- 2 bytes

   -- basic    VehicleBasic,
   size        VehicleSize,          -- 3 bytes

   -- Part II, sent as required
   -- Part II,
   safetyExt   VehicleSafetyExtension OPTIONAL,
   status      VehicleStatus          OPTIONAL,
   ... -- # LOCAL_CONTENT
   }
```

Again, the reader must seek out the Part II content to determine what it allows. Below is the critical two elements of the Part II definition (the VehicleSafetyExtension and VehicleStatus elements) taken from the standard itself, which allows a sequence of possible items and is quite lengthy. Refer to the standard for the precise definition of all these additional data elements.

```
VehicleSafetyExtension ::= SEQUENCE {
   events             EventFlags     OPTIONAL,
   pathHistory        PathHistory    OPTIONAL,
   pathPrediction     PathPrediction OPTIONAL,
   theRTCM            RTCMPackage    OPTIONAL,
   ... -- # LOCAL_CONTENT
   }
```

and

```
VehicleStatus ::= SEQUENCE {
   lights          ExteriorLights OPTIONAL,                -- Exterior Lights
   lightBar        LightbarInUse  OPTIONAL,                -- PS Lights

   wipers   SEQUENCE {
         statusFront    WiperStatusFront,
         rateFront      WiperRate,
         statusRear     WiperStatusRear       OPTIONAL,
         rateRear       WiperRate             OPTIONAL
         } OPTIONAL,                                       -- Wipers

   brakeStatus  BrakeSystemStatus OPTIONAL,
```

```
                -- 2 bytes with the following in it:
                --   wheelBrakes        BrakeAppliedStatus,
                --                      -x- 4 bits
                --   traction           TractionControlState,
                --                      -x- 2 bits
                --   abs                AntiLockBrakeStatus,
                --                      -x- 2 bits
                --   scs                StabilityControlStatus,
                --                      -x- 2 bits
                --   brakeBoost         BrakeBoostApplied,
                --                      -x- 2 bits
                --   spareBits
                --                      -x- 4 bits
                -- Note that is present in BSM Part I
                                                        -- Braking Data
brakePressure   BrakeAppliedPressure  OPTIONAL,         -- Braking Pressure
roadFriction    CoefficientOfFriction OPTIONAL,         -- Roadway Friction


sunData         SunSensor             OPTIONAL,         -- Sun Sensor
rainData        RainSensor            OPTIONAL,         -- Rain Sensor
airTemp         AmbientAirTemperature OPTIONAL,         -- Air Temperature
airPres         AmbientAirPressure    OPTIONAL,         -- Air Pressure

steering   SEQUENCE {
      angle      SteeringWheelAngle,
      confidence SteeringWheelAngleConfidence   OPTIONAL,
      rate       SteeringWheelAngleRateOfChange OPTIONAL,
      wheels     DrivingWheelAngle              OPTIONAL
      } OPTIONAL,                                       -- steering data

accelSets  SEQUENCE {
      accel4way       AccelerationSet4Way           OPTIONAL,
      vertAccelThres  VerticalAccelerationThreshold OPTIONAL,
                                                    -- Wheel Exceeded point
      yawRateCon      YawRateConfidence             OPTIONAL,
                                                     -- Yaw Rate Confidence
      hozAccelCon     AccelerationConfidence        OPTIONAL,
                                                     -- Acceleration Confidence
      confidenceSet   ConfidenceSet                 OPTIONAL
                                                    -- general ConfidenceSet
      } OPTIONAL,


object     SEQUENCE {
      obDist          ObstacleDistance,          -- Obstacle Distance
      obDirect        ObstacleDirection,         -- Obstacle Direction
      dateTime        DDateTime                  -- time detected
      } OPTIONAL,                                       -- detected Obstacle data



fullPos         FullPositionVector OPTIONAL,      -- complete set of time and
                                                  -- position, speed, heading
speedHeadC      SpeedandHeadingandThrottleConfidence OPTIONAL,
speedC          SpeedConfidence OPTIONAL,

vehicleData   SEQUENCE {
      height        VehicleHeight,
      bumpers       BumperHeights,
      mass          VehicleMass,
      trailerWeight TrailerWeight,
      type          VehicleType
      -- values for width and length are sent in BSM Part I as well.
      } OPTIONAL,                                       -- vehicle data
```

```
      vehicleIdent    VehicleIdent OPTIONAL,                    -- comm vehicle data

      j1939data       J1939data OPTIONAL,           -- Various SAE J1938 data items

      weatherReport SEQUENCE {
            isRaining         NTCIP.EssPrecipYesNo,
            rainRate          NTCIP.EssPrecipRate       OPTIONAL,
            precipSituation   NTCIP.EssPrecipSituation  OPTIONAL,
            solarRadiation    NTCIP.EssSolarRadiation   OPTIONAL,
            friction          NTCIP.EssMobileFriction   OPTIONAL
            } OPTIONAL,                                           -- local weather data

      gpsStatus       GPSstatus          OPTIONAL,              -- vehicle's GPS

      ... -- # LOCAL_CONTENT OPTIONAL,
      }
```

We have not implemented all of these Part II optional elements in the code examples of this guide. Rather, a very much reduced set has been implemented to show how such items are encoded in a few examples.[19] A reduced definition of the ASN for this set is provided below.

```
   -- DF_VehicleStatus (Desc Name) Record 57
   -- reduced for example use
   VehicleStatus ::=  SEQUENCE {
      -- data which follows must still fit within any message size limits

      lights          ExteriorLights OPTIONAL,                -- Exterior Lights
      lightBar        LightbarInUse  OPTIONAL,                -- PS Lights

      wipers   SEQUENCE {
            statusFront    WiperStatusFront,
            rateFront      WiperRate,
            statusRear     WiperStatusRear       OPTIONAL,
            rateRear       WiperRate             OPTIONAL
            } OPTIONAL,                                       -- Wipers

      -- Rest removed from listing
      ... -- # LOCAL_CONTENT OPTIONAL,
      }
```

The reader can consult the standard itself, or the provided code base listings (in Annex C) for complete definitions, in ASN form, for additional elements defined in the above code.

In routine use, the Basic Safety Message is created anew every 100 mSec in each sending device, using the then most current data. Additional data may be added based on an on-board determination of events which are taking place (such as hard braking which would cause the event flags data element to also be sent). A recommended duration is suggested here of 500 mSec or 5 transmissions.[20] In this way, additional sent content can be gracefully removed without blocking the processing flow of the initiating process, but other approaches are possible as well. At the same time a receiving socket will be getting BSMs broadcast from other nearby devices at a similar rate. These devices are not synchronized in any manner (other than having generally the same time of day), so the arrival rate represents a stochastic process which triggers action in a classic data flow model. A wide variety of process approaches have been proposed for handling incoming BSM messages, all of which attempt to classify and order the incoming message to known vehicle tracks,

[19] The typical deployment would follow this same process, removing any ASN constructs or data types they did not care to support, and adding any unique locally defined content (in such a way that it did not interfere with definitions established by the standard itself). Additional details of how this process was undertaken to create the guide, ASN and XML used can be found in Annex C, along with a complete schema listing.

[20] Such a recommendation, appearing only in this guide, has no normative value or official standing in the SAE committee process. Consult the SAE committee for definitive advice.

newly detected vehicles, and various classes of possible threats. This application layer processing is out of scope for this guide, and we restrict ourselves to simply recovering and decoding the messages which are received.

Now that the definition of the Basic Safety Message has been provided, we can turn to the process of encoding and decoding information using it.

3.5 Representative Encoding of the Message

In this subsection we outline the actual process of using and encoding each data element found in the message. It is strongly suggested that this subsection be read first, before the other applications, as illustrative details are not repeated in subsequent sections once that type of data processing has been covered. As each data type is handled, make reference to section 6. which handles additional pragmatic details of the *basic type* from which it is made up. For example, the first data element to be covered, `DSRCmsgID`, is in fact an 8 bit integer base type. Integers as a basic type are discussed in subsection 6.6. The rules apply to all integers, regardless of what they are formally called in the ASN of the standard.

In this subsection, each defined type of data used in the message (`DSRCmsgID`, `DSecond`, `Latitude`, `VehicleSize`, etc.) is discussed in turn as it relates to constructing the message at hand. However, additional details on encoding and decoding each specific base type of data in ASN (integers, bytes, strings, array of bytes, structures, etc.) can be found in section 6. Section 6. applies to all the DSRC safety messages[21], and is intended as a general reference to teach how various base types of ASN are handled in ASN DER encoding, and how each type is kept as a C++ data type or as a complex structure.

It is very important at this point to recall that the ASN.1 of the standard describes the structure and ordering of the message *over the air* between systems. It does not describe how the same message is kept in the memory of each system at the sending or receiving end.[22] In fact, this typically differs with the conventions used by each ASN.1 tool vendor. A precise one to one mapping is not possible for a number of reasons, as is illustrated in greater detail in section 6. This is most often seen in the way complex nested structures are treated (using pointers to each part) and the way optionally present elements are treated (typically with some additional feature being needed to determine the presence or absence of the element). This is why the resulting *.h structures, which the vendor's ASN library returns to the user for his own use with application to ASN.1 code, will not map directly to the message being sent. Let us use the memory structure of the BSM to examine this issue in further detail.

The top level structure of the BSM produced by the ASN1c tool is as follows:

```
/* BasicSafetyMessage */
typedef struct BasicSafetyMessage {
    DSRCmsgID_t            msgID;
    BSMblob_t          blob1;
    struct VehicleSafetyExtension *safetyExt    /* OPTIONAL */;
    struct VehicleStatus       *status          /* OPTIONAL */;
    /*
     * This type is extensible,
     * possible extensions are below.
     */

    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _      asn_ctx;
} BasicSafetyMessage_t;
```

[21] In fact any and all ASN messages encoded in the ASN DER style.

[22] These may differ depending on how each side has implemented the message and what choices of ASN.1 tools and libraries and options are selected to be used. Interoperability in this context is defined by the message exchange, not how the message is kept, processed, or used.

Compare each line to the ASN representation given in the last section. A line of ASN.1 such as:

```
msgID            DSRCmsgID,              -- 1 byte
```

Becomes the following in the *.h files:

```
DSRCmsgID_t    msgID;
```

The instance `msgId`, of the defined type `DSRCmsgID` in the ASN.1 has become in the language C the data element named `msgID`, of type `DSRCmsgID_t` in the *.h file (the `_t` added to the end seems to be this tool's way of indicating a type defined by the users ASN.1). Note that comments are dropped in the process, and there are supporting files called *DSRCmsgID.h* and *DSRCmsgID.c* in the resulting library of files. In fact the element `DSRCmsgID` happens to be an extensible enumeration, which in this tool is modeled as a `long` integer. Each of the possible legal values of the item (the assigned numeric values for messages) is represented as a named item in the enumeration present in the C language. Thus the code in the ASN of:

```
-- DE_DSRC MessageID (Desc Name)
DSRCmsgID ::= ENUMERATED {
   reserved                     (0),
   alaCarteMessage              (1),
   basicSafetyMessage           (2),      etc.
```

becomes in C the following:

```
typedef enum DSRCmsgID {
    DSRCmsgID_reserved           = 0,
    DSRCmsgID_alaCarteMessage    = 1,
    DSRCmsgID_basicSafetyMessage = 2,      etc.
```

This type of naming is found in most every vendor tool in more or less the same style. The instance names become the names of variables in structures, and the formally defined type definition names become the names of globally defined types with supporting C and H files to perform the encoding. Some *name mangling* can still occur. Typically this is because ASN.1 allows the use of the dash (-) character in names, while C allows the underscore character (_). In rare cases this can result in collision problems.[23] As a merged rule of both ASN.1 and the style of XML used throughout ITS, formal types are always given with a first character in upper case, while instances of those types always begin with a lower case. This can cause confusion with other coding conventions, but is consistently used throughout. It should be noted that a similar style is used in the expression of the XML schema objects defined in the standard as well.

In order to begin filling in the data types for such a message, it is necessary to allocate a copy of the top level structure into memory. This is done with the normal C allocation calls as follows (the created instance is called `myType` in the example). We then deal with how such a structure is used in filling out the message. Note that `BasicSafetyMessage_t` is defined in a file called *BasicSafetyMessage.h* which links to the rest of the internal elements of this message. All of the other defined messages are allocated in a similar fashion.

```
BasicSafetyMessage_t  *myType;
      // pointer to the msg struc
myType = (BasicSafetyMessage_t *)calloc(1,sizeof *myType);
      // allocate one copy in memory
```

Returning to the BSM message in its memory form, note that we have a mixture of pointers to things (`safetyExt` and `status`) and of basic data types (`msgID` and `blob1`) stored inline in the structure. While the *.h structure makes it obvious that `safetyExt` and `status` (the Part II items) are using pointers (due to the use of *xxx), one must trace out the data elements `msgID` and `blob1` and look at their definitions to determine this. We also have some items inserted by the tool to

[23] At present the only known cases of this all occur in using the revision 2.1 of the TMDD standard.

allow later extensible objects (the … in the ASN source code) to be added which we ignore for now. None of the items is immediately discernable as to its type due to the ITS standard practice of always having a proper Type Definition name for elements.[24] Each of these defined types has its own library code module and *.h files produced by the ASN tool we have chosen to use.[25] In addition, there are a number of basic modules that deal with the primitive types. Complex types call or include lower types, producing a nested collection of files that work together to represent the entire defined message. This is typical for such libraries.

However by tracking down some of these (denoted in the comments at the right) we can determine that:

```
/* BasicSafetyMessage */
typedef struct BasicSafetyMessage {
    DSRCmsgID_t    msgID;                       // A 1 byte instance
    BSMblob_t  blob1;                           // A 38 byte instance
    struct VehicleSafetyExtension *safetyExt;   // A pointer
    struct VehicleStatus    *status;            // A pointer
    asn_struct_ctx_t _asn_ctx;                  // ignore, not used
} BasicSafetyMessage_t;
```

Now it becomes clearer when we have a pointer type or complex type, and where we have a simple data type can be determined by looking at the relevant types (the declared types, all starting with a capital letter in the C listing). In our example program, setting the values for one of the simple types is done in the normal C way, shown below.

```
// start to fill in the data here
myType->msgID = DSRCmsgID_basicSafetyMessage;
       // assign a constant to this simple integer
       // note that this constant is provied by the enumerated types
```

In a similar fashion we can simply assign some representative (but invalid) data items to the “blob” with a copy operation, although more typically the structured data defined to be inside the blob must be more artfully constructed.

```
// do the data blob here, fill it with a set of increasing values
for (j=0; j < 38; j++) {
    myType->blob1.buf[j] = j;
}
```

We must also first allocate additional memory for the pointer types before any value can be set for them. In the case of the DSRC message set standard, the maximum length of any string or data value is always known because the lengths of all defined types are always constrained to known sizes. This is not true in some other areas of ITS standards. Further, all strings in ASN (when encoded in DER) use the Type-Length-Value (T-L-V) method to denote the length, not a control character (such as \n) to terminate strings. Allocating pointers to suitable space is done in the normal C manner, as in this simple example for a four byte temp ID[26] value:

```
myType->id.buf = (uint8_t *)calloc(1,4);    // allocate 4 bytes
assert (myType->id.buf);                    // test for errors
myType->id.size = 4;
```

[24] The alternative style would be to call items like `msgID` an integer directly, and therefore we would clearly see the primitive type used. However, this style of use prevents and discourages data element reuse, a prime goal of the ITS standards. If used, it becomes impossible to readily tell when a data concept is being (or could be) reused (as they are all just called things like *integer*) and each individual instance can be changed allowing coding errors to occur and remain undetected.

[25] If you do not have a copy of the code base in front of you at this time, refer to Annex B where these files are listed. The important item to note is that that a great number of small files (76 of them for just the BSM) are automatically created by the ASN.1 tool (when provided with the ASN source code) which the end application code will use or call to handle the actual encoding work. Each file matches the name of a data element type, such as *WiperStatusFront.c*, that it handles.

[26] In the blob, this data element is part of the octet set (and therefore need not be allocated separately), but in other data frames this data element is encoded separately (and as an optional element) and could be handled in this way.

A better example would be to use the size declared for the object itself in the allocation,. This is the preferred approach so that iterative changes in the size of your objects does not impact the reliability of your code. Note that this method works even when the object is of variable message size because its memory footprint is always of a finite and known size.[27]

```
Better expample here using a sizeof() cast
myType->xxx.buf = (uint8_t *)calloc(1, sizeof(xxx)); // allocate bytes
assert (myType->id.buf);
myType->id.size = sizeof(xxx);
```

Let us consider the `BSMblob_t` structure a bit further. The octet string (used in the blob and elsewhere) is one of the basic (primitive) types implemented by the ASN.1 library routines, which is (in the *OCTET_STRING.h* file[28]) defined pretty much as one would expect, as:

```
typedef struct OCTET_STRING {
    uint8_t *buf;    /* Buffer with consecutive OCTET_STRING bits */
    int size;        /* Size of the buffer */

    asn_struct_ctx_t _asn_ctx;
                     /* Parsing across buffer boundaries */
                     /* which is not used in the DSRC work */
} OCTET_STRING_t;
```

Once each buffer has been allocated, it may be assigned values in any way the user desires. In the prior code fragment we simply loaded 38 bytes of the `blob1` or `BSMblob` with some random (and therefore meaningless) count values as an example. Normally this particular data element would be built up from GPS positions and vehicle motion readings (an example of which will be given later).

Loading a data blob is easy, but creating it out of collections of packed data items may not be. In the case of the blob defined here, we have a large number of other data items that must be built up into the data stream in the correct order. Thankfully this is one time where a structure definition of the required elements is in fact equal to the stream of octets that is needed. Unions can be used at such times with care. At other times (especially when bytes and bit ordering defined by the processor do not match that defined by ASN rules and may be reversed) this may not be so. The developer must be very certain of the size of variables used in the target system when doing this sort of thing. Here is a simple structure which defines the packed order of the data elements to be sent in the blob. Compare this with the definition of the `BSMblob` given before. Notice also that the ASN library file *BSMblob.h* does not contain any of these details, as the rules are not reflected in ASN itself (one must consult the actual SAE standard).

```
/* Blob1 structure */
typedef struct Blob1  {
    // Part I Content, sent at all times

    BYTE   msgCnt       // Message count                    -- 1 byte
    long   id           // A temp ID field                  -- 4 bytes
    SHORT  secMark      // the current second               -- 2 bytes

    // pos      PositionLocal3D,
    long   lat;         // lat         Latitude,            -- 4 bytes
    long   longi;       // long        Longitude,           -- 4 bytes
                        // note: pick a different name to avoid collisions
    SHORT  elev;        // elev        Elevation,           -- 2 bytes
    long   accuracy;    // accuracy    PositionalAccuracy,  -- 4 bytes
```

[27] Variable length objects are either fully allocated (when declared inline) or a pointer is used when they are optional. .

[28] In the tool we are using (ASN1c) the filenames of all the primitive types are in all caps as in: *OCTET_STRING.h* or *INTEGER.h*, etc. In this particular library implementation, these files are *canned* and do not change with the supplied ASN.1, whereas all the other files represent implementations using these files and given the names provided by the source ASN.1 (in this case the DSRC standard).

```
//  motion   Motion,
SHORT  speed;    // speed       Speed and Transmission -- 2 bytes
SHORT  heading;  // heading     Heading,               -- 2 bytes
SHORT  angle;    // steering    Angle,                 -- 1 byte

// accelSet    AccelerationSet4Way,  -- accel set (four way) 7 bytes
SHORT  accelLong;
SHORT  accelLatSet;
SHORT  accelVert;
BYTE   accelYaw;

// -- control  Control,
SHORT  brakes;   // brakes      BrakeSystemStatus,     -- 2 bytes
// see std for bit make up

// -- basic    VehicleBasic,
// size        VehicleSize,          -- 6 bytes
// see std for bit make up (width and length as 3 byte values)
BYTE   size1;    // size (width)
BYTE   size2;    // size (width)
BYTE   size3;    // size (width)
BYTE   size4;    // size (size)
BYTE   size5;    // size (size)
BYTE   size6;    // size (size)
} Blob1_t;
```

The individual elements of this structure can be assigned in the normal way to the blob instance found in the BSM. The source code provides examples of this; see the *BLOB1.cpp* file and note how in the blob dialog the actual blob's bytes are loaded and unloaded in the two functions *PackIt()* and *UnPackIt()*. It should be noted that this style of coding is more or less circumventing the features provided by using ASN and was a conscious design decision by the DSRC committee to reduce the transmitted payload in this case.[29] It can be found in a few other places in the standard, and the encoding rules discussed here can be applied to these other cases. It is also important to note that no content checking occurs by the ASN.1 library for such blobs (only the length constant is checked). Any additional checking becomes the responsibility of the application layer to perform.

Optional elements defined elsewhere can be filled in or not, depending on the need to send them in the message (driven by the application in this instance). If an element is not allocated or provided with a value, it is not sent. Most simple elements types have defined values that can be used when the data is not available or out of range in some way. Note that where an element is defined as used in a blob it must be sent to preserve the expected blob length.

Frankly, the BSM message, having only four data items at the top level, represents a more trivial example of the encoding effort needed for our first application. Here we have shown how to encode two items directly into memory, and discussed the details of the two other items that use pointers. The next application, that of a *RoadSide Alert*, is a message comprised of many separate data items and provides a broader set of encoding examples. We now turn our attention to how to use the memory image to create an actual message.

Once this building process has been done for all relevant data items in the message, the memory structure of the message can be used in a variety of ways. The most typical of these is to encode it into the sequence of bytes to be turned over to the IEEE 1609 layer to be sent (i.e., the final

[29] The general rationale for creating such octet blobs is that many small and well defined objects (such as a few bytes with constant content and known meanings) can be encoded with only three additional bytes in the payload. If the same content was encoded such that each individual element had the T-L-V encoding of ASN DER, then an additional 2~3 bytes per data item would be needed. However, such a design choice places some of the message decoding logic into the application layer (typically causing the received message to be processed a second time, after the actual T-L-V has been stripped off by a common ASN library, to strip out the blob data), which can be confusing from a code maintenance perspective.

message payload which is sent to the next layer for broadcast over the WSM socket). The socket layer then adds the IP datagram layers to the payload. The serialized routines provided by the ASN.1 library handles all the payload encoding details (include dealing with stuffing optional content) and returns a stream of payload bytes:

```
// a global to hold the PAYLOAD data
#define MAX_MSG_SIZE 100
BYTE msgArray[MAX_MSG_SIZE];
                  // an array of bytes to hold the BER encoding
BYTE *theMsg;    // a ptr to it
theMsg = &msgArray[0];

// render it out into the BER format
i = DSRC_serializer( myType, theMsg);
fprintf (stdout, "Created a BSM message payload with: %d bytes.\n", i);
```

The critical function being:

```
ssize_t DSRC_serializer(
   BasicSafetyMessage_t  *theMsg,    // the ptr to the struc
   BYTE *theBuffer )                 // the ptr to the buff to fill
```

Another useful process is to render the same message out in its XML form, although the XML produced by this tool is not 100% in accordance with that established by the standard, nor the styles found in ITS in general. Notably, enumerations are not handled correctly as unions of known strings and integers as defined in the DSRC message set standard.

```
xer_fprint(stdout, &asn_DEF_BasicSafetyMessage, myType);
```

Feeding a stream of data into the tool can also be used to decode a message payload recovered from the lower layers into this same structure. Here is the critical function used:

```
static void *data_decode_from_file(
   asn_TYPE_descriptor_t *asnTypeOfPDU,
   FILE *file,
   const char *name,
   ssize_t suggested_bufsize,
   int first_pdu);
```

As you can see, the library routines more or less hide the details of encoding (and also whatever constraint checking may be present) from the user. The ASN library provided *.h files for the message become the key media of exchange between the user's application program and the lower library routines. This is further illustrated in the figure below where the use and duties of each layer is clearly shown in the process of developing a BSM for transmission. A similar image could be constructed for any of the other messages.

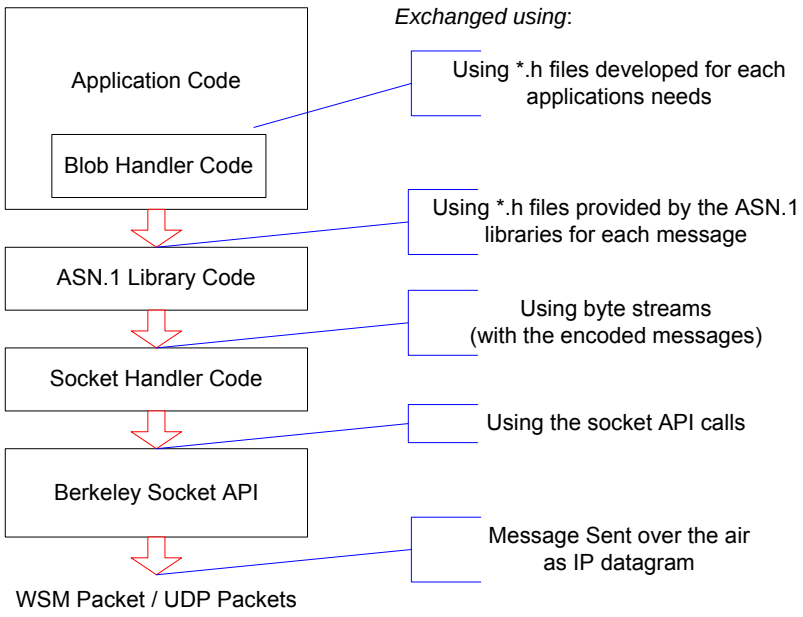

Figure 3 – Protocol Stack Used in the BSM Examples

The message itself, once encoded as a complete payload, is handed over to the WSM network service layer, along with some basic control information (also called the management information plane in some protocols) to be transmitted. The WSM layer handles the details of the rest of the communications stack and sending the message out over the airlink. Receiving vehicles, also running a WSM layer and protocol stack, get the message and deliver it to applications who have registered to receive it. Messages received with errors or with other problems are typically dropped in the WSM layer and not forwarded to the application. The precise mechanical details of how this works (as well as what specific features that may be present) remains up to each implementation at this time. In other words, the WSM and the IEEE 1609 specification does not yet provide a standardized API for how these services are exposed.

In this guide, where the code base is intended to be used on an MS Windows PC for educational purposes, we have solved this by making use of UDP packets broadcast over TCP/IP. We have avoided using any of the unique MS Windows protocols or services. Further, we have chosen to use a Berkeley style socket for the API, believing that this choice aids deployments which will likely port the code examples to a machine running UNIX or an embedded system where this type of interface is common. Messages are handed over to the socket (using an industry standard API implemented by Microsoft) and transmitted to the user's local subnet. A simple *listener* has been installed on the appropriate port to recover messages when sent, and to forward them to the braking application (forming a very primitive message dispatcher). With this approach, simple network clusters of devices can be created to exchange messages. Refer to section 8. for additional technical details regarding how the lower layers have been implemented on the PC platform.

3.6 Examples of Well-Formed Messages

At this point we will construct some well-formed messages to examine in greater detail the resulting stream of bytes produced and used in the exchange. We begin with as simple a valid message as practical, then add additional details as familiarity with the encoding grows. Let us begin with a message containing simple values for the two required items defined in the Part I section of the message, a *zero version* if you will:

Table 3 – Data Elements in a Very Simple but Valid BSM Message

Element Name	Simple or Null Value	Byte Length	Hex Values	Comments
msgID	2	1	0x02	Defined by std
blob1	All Zeros	38	0x00,00,00,00, etc.	Arbitrary

The above data was created using the code fragments given previously (in the tool provided in the supporting subsections; see section 8. for the source code). The resulting message, once encoded in BER-DER form, becomes the following stream of 45 bytes. The red box indicates one of the two T-L-V structures present in the message (used to hold the `msgID` element).

```
0x30 0x2B 0x80 0x01 0x02 0x81 0x26 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00
```

A few moments can now be spent on some analysis of the internal structures found in this data. Recall also that these details are also dealt with in section 6., where the specifics of ASN encoding are covered. If the reader has access to an ASN decoder or viewer tool, it may be useful to load this same message into that tool while reviewing this section. The above information can be found in the supporting data file *BSM_1.dat*. If you are using the DSRC tool to create this message, you would have a dialog box that looks like this:

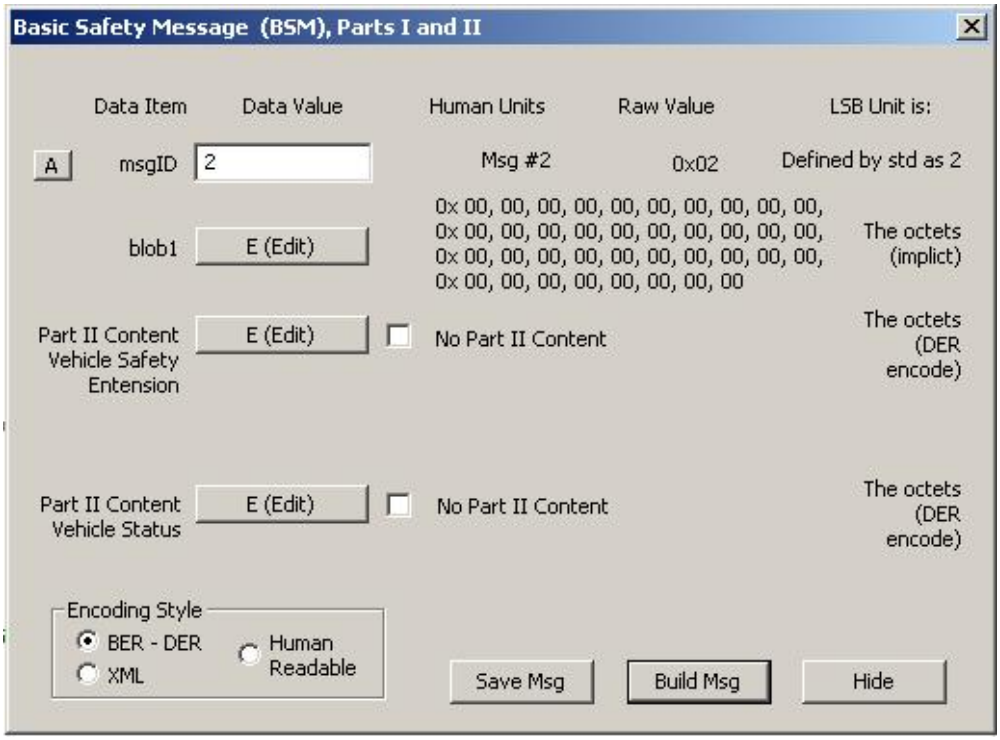

Figure 4 – The Data Elements Displayed in the DSRC Tool, BSM Dialog

The first observation a person new to ASN decoding is likely to make is that this message does not seem to start with any T-L-V that has `0x02` (the value for the first element, `msgID`) as one might expect. Rather the value `02` occurs in the fifth byte position (the last byte of the first red box). It is preceded by a 01 (which you may rightly assume is the length set to one byte) and a `0x80` which in fact is the first basic tag inside the message.[30]

However, occurring before either of these are two bytes consisting of `0x30` and `0x2B`. These two bytes (always present in such a message) serve to announce that the first tag found is in fact a sequence[31] (tag `0x30`) and that the length is `0x2B` additional bytes (43 in base ten). These are the T and L of the topmost enclosing structure. The V is the remaining bytes representing the sequence value contents (in this case the TLVs for `msgID` and `blob1`.). Thus we see our first example of the nesting of T-L-V structures in a message.

After the first two bytes, we typically see a reoccurring pattern of two T-L-V groups in any ASN message. The first one is denoted here by a red box, while the last one, representing the octet blob1, is not shown in red. Note that there is no other outer *wrapper* or a count of the size of the entire message present here (there is however one in the enclosing IP datagram in which it is sent). There is no final character, such as a \n, present either.[32] The first T-L-V declares that the message ends after `0x2B` or 43 additional bytes, and that is all there is to it. The above stream of bytes is what the lower layers send and receive and represents the complete DSRC message.

The above exercise in decoding the byte stream can be made much easier using COTS tools to display the structures present in the ASN. Below is the image produced by the ASN.1 viewer tool ASN1c (see sections 6. and 7. for further details) of this same data.

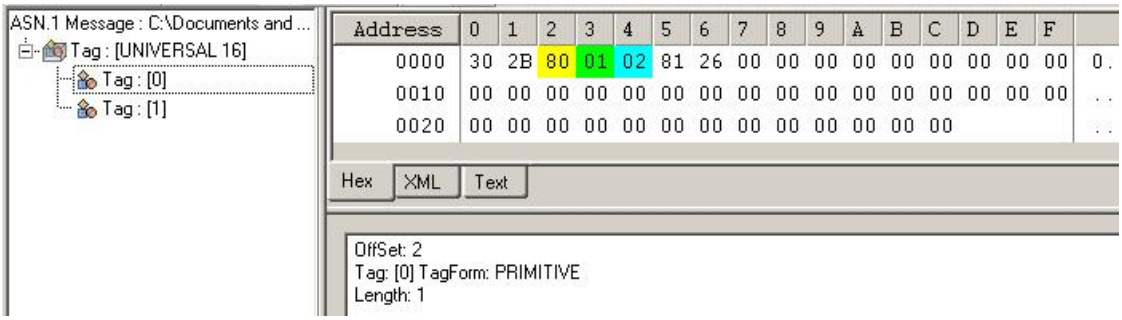

Figure 5 – The Data Elements Displayed in an ASN Decoder Tool

The highlighted three bytes (80, 01, 02) represent the range taken up by the object "Tag: [0]" shown highlighted in the left panel. The lower panel shows that this object is found two bytes into the message, is of a primitive type, and the value being one byte in length. Note that the T-L-V objects are not named in any meaningful way, i.e., (Tag[0], Tag[1], etc.). In other words, this tool is decoding the ASN stream of bytes without any knowledge of what our ASN is encoded to mean or represent. In such an environment the types and lengths of the data elements can be determined, but not the various constraint relationships or any of the assigned instance names. If we load the same ASN source file used to produce this encoding, then the tool can further annotate and add these details as shown below.

[30] The astute reader may notice that there is a lot of 0x80, 81, 82 values occurring in such ASN, as these represent the assigned local tags values of 1, 2, 3, etc. Upper and lower tag nibbles in tag assignment are covered further in section 6., but in general a user need not be aware of these rules.

[31] And therefore this item will nest any number of additional defined elements (as per our ASN.1 source) inside of it.

[32] In fact the value zero can occur in the byte values of the data stream as the prior example shows in multiple places. When used in ASN, it does not imply an end of the string delimiter as it would in an C string.

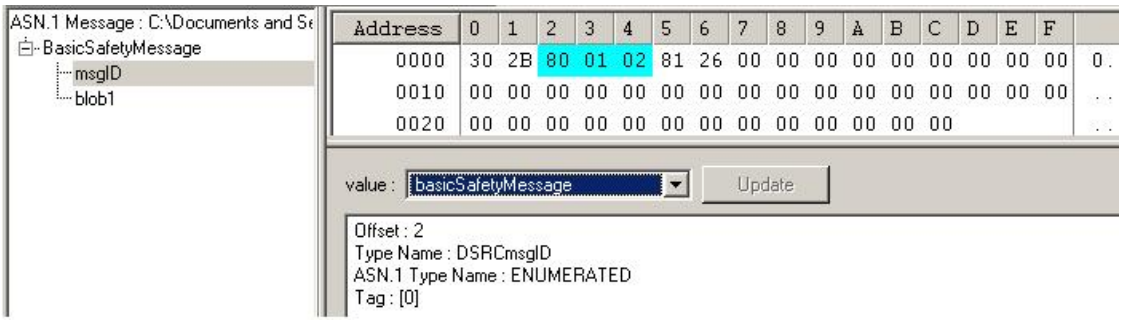

Figure 6 – The Data Elements Displayed in an ASN Decoder Tool, with Source ASN

Now the instance names of items (such as `msgID` for the first tag) are preset. We also see that the highlighted tag (`msgID`, or Tag[0]) is of the type `DSRCmsgID` which is an enumeration. There is no Part II content shown because there is no Part II content present in the message. Observe that the `blob1` element remains a stream of bytes, and that no further decoding occurs for such objects.

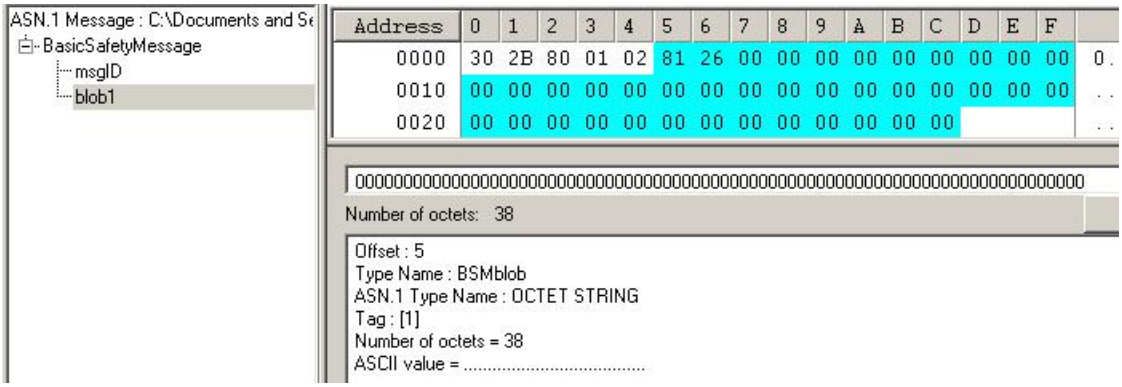

Figure 7 – The blob1 Data Elements Displayed in an ASN Decoder Tool, with Source ASN

This is not to imply that there is no further structure in the blob, only that the ASN rules are not aware of it (and therefore do not detect and decode it).

If you are reading this document and have access to the above tools (they can easily be downloaded, see section 6.) now would be a very good opportunity to take a break and play with these tools yourself. The remainder of the document makes increasingly heavy use of both tools to *create*, *decode*, and *display* the binary contents of well-formed messages.

Let us now consider some additional message payload values and construct a valid message for these. This can be done using the software provided, or by using the commercial viewer tools provided; the result is the same. However, the handling of the blob data still requires extra effort to assemble the proper sequence of bytes, which is made easier by the use of the provided tool.

Adding some arbitrary but reasonable data to the blob area we then have the below values. Keep in mind that in this case there are no additional bytes being used for extensibility, nor is any of the optional Part II content present.

Table 4 – Data Elements in a Valid BSM Message

Element Name	Human Value	Byte Length	Hex Value	Comments
msgID	2	1	0x02	Defined by std
blob1		38		Built up of:
msgCnt	1	1	0x01	Arbitrary
secMark	60000	2	0xEA, 60	Arbitrary
id	32,33,34,35	4	0x20, 21, 22, 23	Arbitrary
lat	35 deg north	4	0x14DC9380	1/10th micro degree
long	120 deg west	4	0x47868C00	1/10th micro degree
elev	1000 meters	2	0x2710	
accuracy	Perfect Accuracy	4	00, 00, 00, 00	
speed	50.00 m/s	2	0x49C4	in 0.02 m/s, drive
heading	Due South	2	0x3840	in 0.0125 deg
angle	Ahead	1	0x0	in 1.5 deg
accelSet	None	7	0000,0000,00,0000	None, and no yaw
brakes	All off	2	0A,A0	trac/abs/scs present
size (width)	220 cm	3 (½ of 3)	0DC	about 6 ft
size (length)	670 cm	3 (½ of 3)	29E	about 22 ft

If you were using the DSRC guide tool to create the above data, the dialog of the blob data would be as follows:

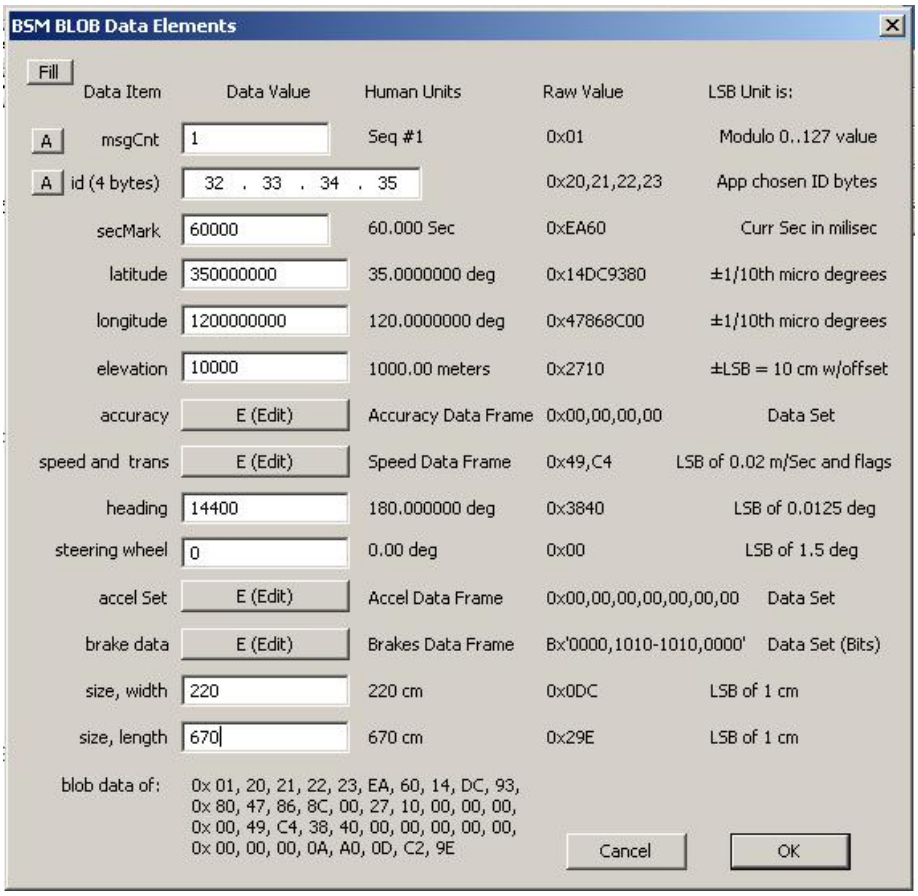

Figure 8 – The Data Elements Displayed in the DSRC Tool, Data blob Dialog

This data is represented by the stream of bytes shown below and in the image further below (again shown with the tool decoding it at the left). Note that the complete data blob contains 38 octets, representing the elements in the above array:

```
0x..01, 20, 21, 22, 23, EA, 60, 14, DC, 93,
    80, 47, 86, 8C, 00, 27, 10, 00, 00, 00,
    00, 49, C4, 38, 40, 00, 00, 00, 00, 00,
    00, 00, 00, 0A, A0, 0D, C2, 9E
```

This stream of octets is itself a complex and compound structure (in that `speed, accuracy, size` and `accelSet` are each complex elements made up of further definitions). Note that generally this aligns on nice byte boundaries, the only exception being the last two components which make up the `size` element (`width` and `length` respectively) which are defined as 3 byte values. Again, any nibble manipulation becomes the responsibility of the application level to perform here, not the ASN library. The ASN encoder would, if given the chance to do so, represent the 3 byte values in three bytes, unless the sent value could be represented in one or two bytes. And the memory footprint for the element (see the `size_t` element in the *BSM.h* file of the source code) would likely be cast as a `long` to avoid this issue once the message was decoded.

Encoded as a complete message, the following stream of bytes would be as shown below.

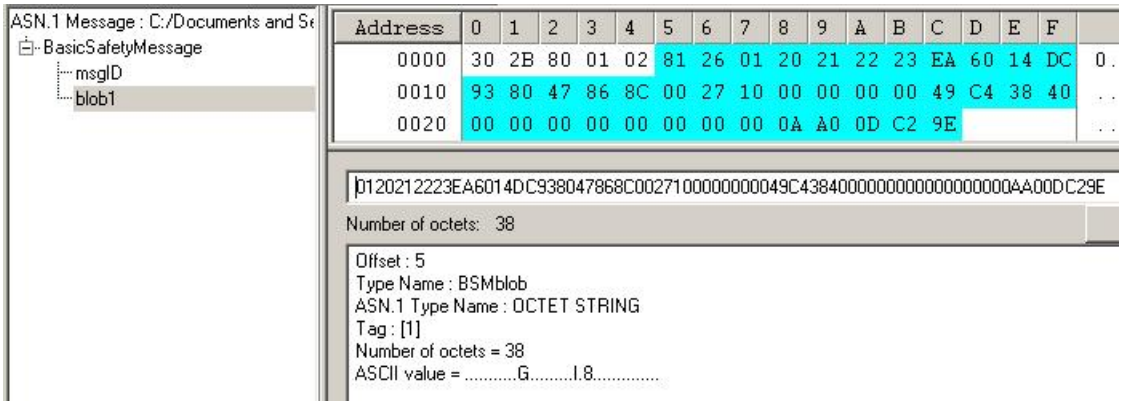

Figure 9 – Valid Data Elements Displayed in an ASN Decoder Tool, with Source ASN

Observe that the second tagged item (the blob) in the ASN contains all of this complex content but that no further internal tagging is to be seen as the structure is defined as simply an octet string (the content is simply run together in the bytes). One can pick out the data from the table in the above representation. In the next application example we will see a multiplicity of data elements each with their own tag values, which is the contrasting style. This example, using an octet as a blob, is typical of a more primitive approach but also saves transmission bandwidth at the expense of some additional decoding effort. Note that such a message can only be used when the structure is not subject to optional content because it depends on the known position of objects inside the blob.

Now let us turn our attention to the legal variants of this message when some additional `OPTIONAL` content is added to the end of the message. This results in an additional T-L-V item for each item that is present at the end, and of course the overall length of the message itself grows, which is reflected in byte 2 (value `0x2B` in the above). If the message were to grow beyond 127 bytes, this length byte would become a two byte value reflecting the size of the enclosed V in the initial sequence structure (refer to section 6. for details).

Adding one of the two `OPTIONAL` Part II elements, the `safetyExt` and the `status` content, we have the below (Figure 10) when there is none of the inner element defined in either present. This shows the two outer tags as empty "shells" to hold the content defined in either of the data frames. In actual practice you would not create such an object unless at least one internal element was defined. The usage textual rules of the standard outline this for the case where a `SEQUENCE` is made up of only `OPTIONAL` internal content. Again, such encoding may be valid ASN but they are not considered valid DSRC messages unless there is also some inner content inside the structure. For the purpose of our educational use, consider it as a starting point to which we will now add the additional internal content represented by the "wipers" content.

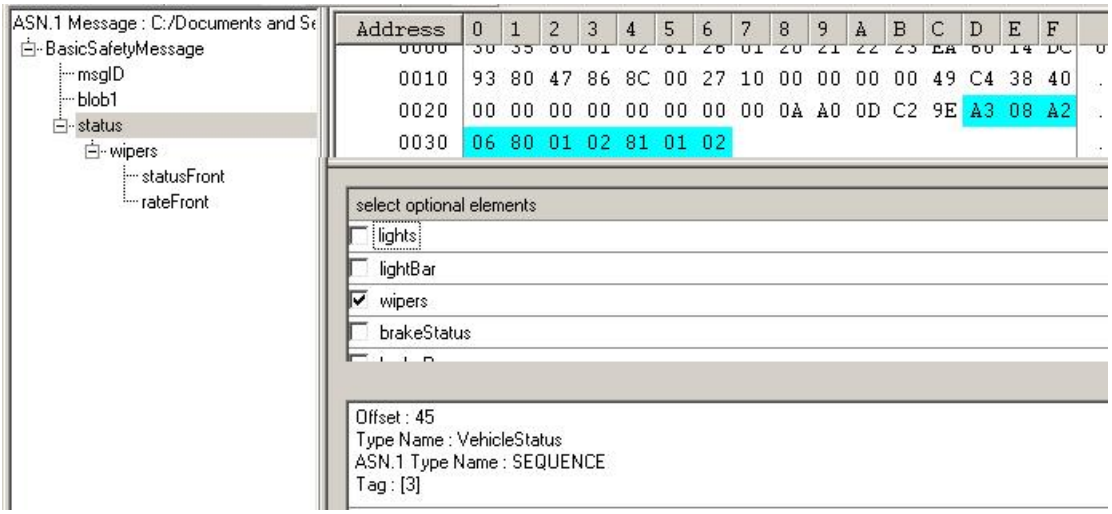

Figure 10 – Valid Data Elements Displayed in an ASN Decoder Tool, with Part II Content

If we were to add the "wipers" data frame (defined as part of our subset for the "status" element of Part II) and then add the data element for the front and the data element rate for the front wiper set to the prior message, we would have the stream of bytes shown below.

The data concept `wipers` itself consists of two required elements: the front wiper status (`statusFront` a byte) and the wiper rate (`rateFront` a byte), and an optional rear side wiper status and rate as well. These optional elements are not present and therefore there is simply no tagging to represent them. If any additional optional elements were present they would appear in the order mandated by the standard within the *wrapper* provided by the T-L-V of the `vehicleStatus`. For instance, the BSM code we have used in these examples (which is reduced from the full standard) allows the `lightBar` and `lights` data elements to precede the `wipers` data. This is shown by empty checkboxes in the tool (displaying what might occur and where) when the enclosing object is highlighted.

Note that the `vehicleStatus` structure encloses the `wipers` structure with 12 bytes and the lines:

```
A3 08 A2 06 80 01 02 81 10 02
```

There are several *sequence* tags being used in this section, reflecting the two layers of structure defined in the message. The first (`0xA3`) represents the `status` element, and the second (`0xA2`) represents the `wipers` element. The numbering assigned to the tags (1, and 2 then 1 and 2) is determined by the number of other preceding elements in the structure at the same level of ASN.1 code. For example, `status` is labeled as the 03 tag (in fact A3 as it is a structure) because it is the 4th element after `msgId`, `blob1`, and three elements may come before it (note that the numbering begins with zero). Note that the tag numbering includes optional elements as well. The process of

numbering is taken care of by the ASN.1 tools, but the creator of the source ASN can specify specific tag values as well. This is how locally defined ASN content avoids colliding with that defined by the national standard. By convention, all such local tags use values from 128 to 255 for their tags, in a process similar to the way enumeration values in the range of 128..255 are often set aside for local use.

The `wipers` data structure, consisting itself of the two inner items (`statusFront` and `rateFront`) is contained in the highlighted lines below (and in the eight bytes of content beginning with byte 47 starting with a value of `0xA2`, highlighted). Note again, the optional checkboxes provided by the tool to indicate that additional content (in this case the two elements relating to the rear wipers) could be added to the `wipers` data structure.

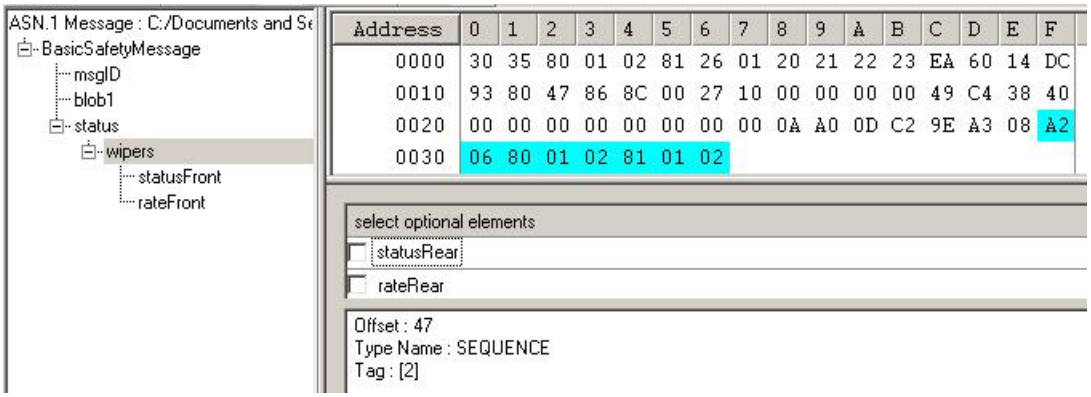

Figure 11 – Valid Data Elements Displayed in an ASN Decoder Tool, with Part II Content, Showing "Wipers"

Observe that the two wiper elements are simply encoded as 80-01-02 and 81-01-02 in each case. In the first of these, the value 2 is an enumeration indicating intermittent use, while in the second it is a value indicating a rate of two sweeps per minute. Note that in encoding and transmission, an enumeration is the same as an integer. Note also that in all of these cases, the simple format of the T-L-V is used to describe and enclose the object regardless of whether it is simple or complex in nature.

As a final example, we create one additional message where the various required data values are all set to the maximum values allowed. The maximum allowed values are shown in the table below along with their hex representations. Note that in some cases the value that is sent is "one past" the range of valid data that is supported by that data element.

Table 5 – Data Elements in a Valid BSM Message, Maximum Values Used

Element Name	Human Value	Byte Length	Hex Value	Comments
msgID	2	1	02	Defined by std
blob1		38		Built up of:
msgCnt	-	1		Arbitrary
secMark	-	2		Arbitrary
id	--	4		Arbitrary
lat	90 deg north	4	35A4E900	1/10th micro degree
long	180 deg west	4	6B49D200	1/10th micro degree
elev	6143.9 meters	2	EFFF	
accuracy	Worst Accuracy	4	7F, 7F, 7F, FF	
speed	163.8 m/s	2	5FFE	in 0.02 m/s
heading	Due North	2	0x7080	In 0.0125 deg
angle	00	1	0x0	In 1.5 deg
accelSet	Extreme	7	07D0,07D0,7F,7FFD	Max values
brakes	All on/Active	2	FFF0	trac/abs/scs on
size (width)	1023 cm	3 (½ of 3)	3FF	about 33.5 ft
size (length)	4095 cm	3 (½ of 3)	FFF	about 134.4 ft

And now the `blob1` becomes the octet string:

```
0x..01, 00, 00, 00, 00, 00, 00, 35, A4, E9,
    00, 6B, 49, D2, 00, EF, FF, 7F, 7F, 7f,
    FF, 5F, FE, 70, 80, 00, 07, D0, 07, D0,
    7F, 7F, FD, FF, F0, 3F, FF, FF
```

With the resulting complete BSM message shown below:

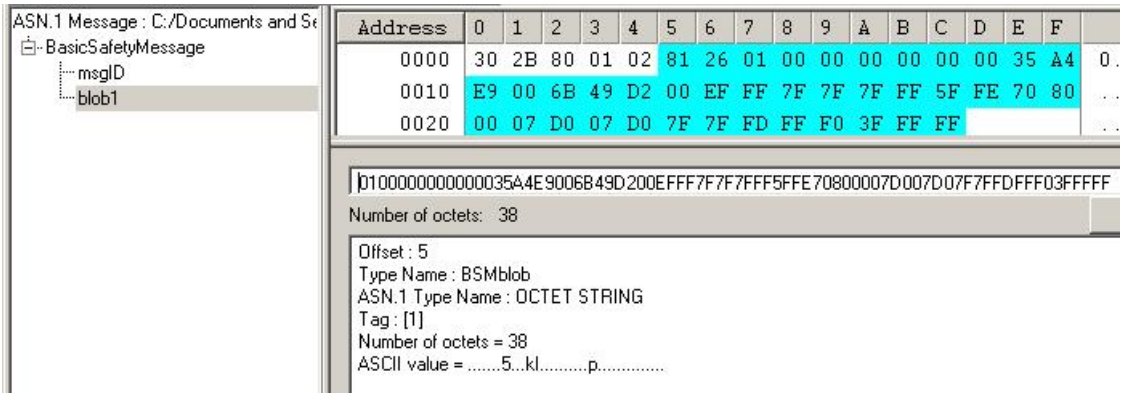

Figure 12 – Valid Data Elements Displayed in an ASN Decoder Tool, Showing blob Content

If this data set was created using the DSRC tool and code examples, the two critical dialogs would be as follows:

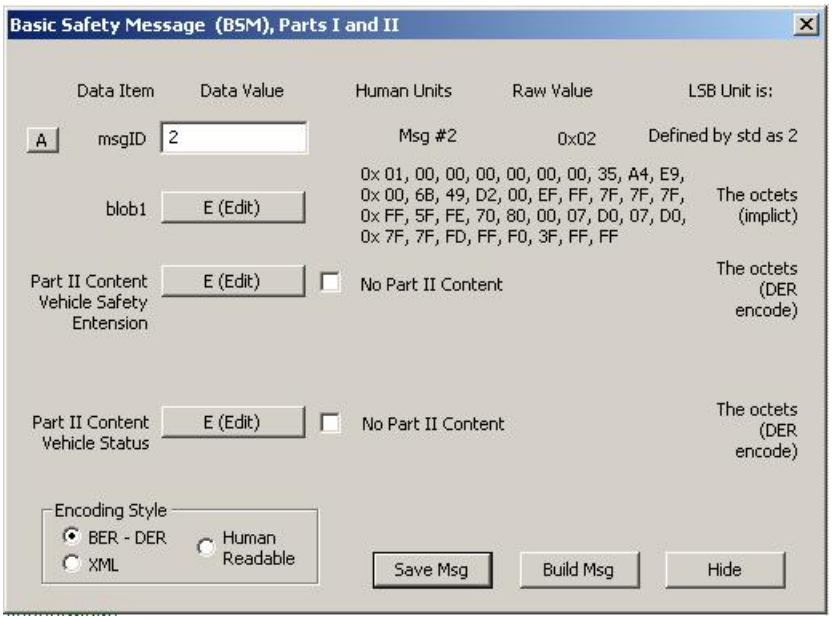

Figure 13 – Top Level BSM Data Shown in the DSRC Tool

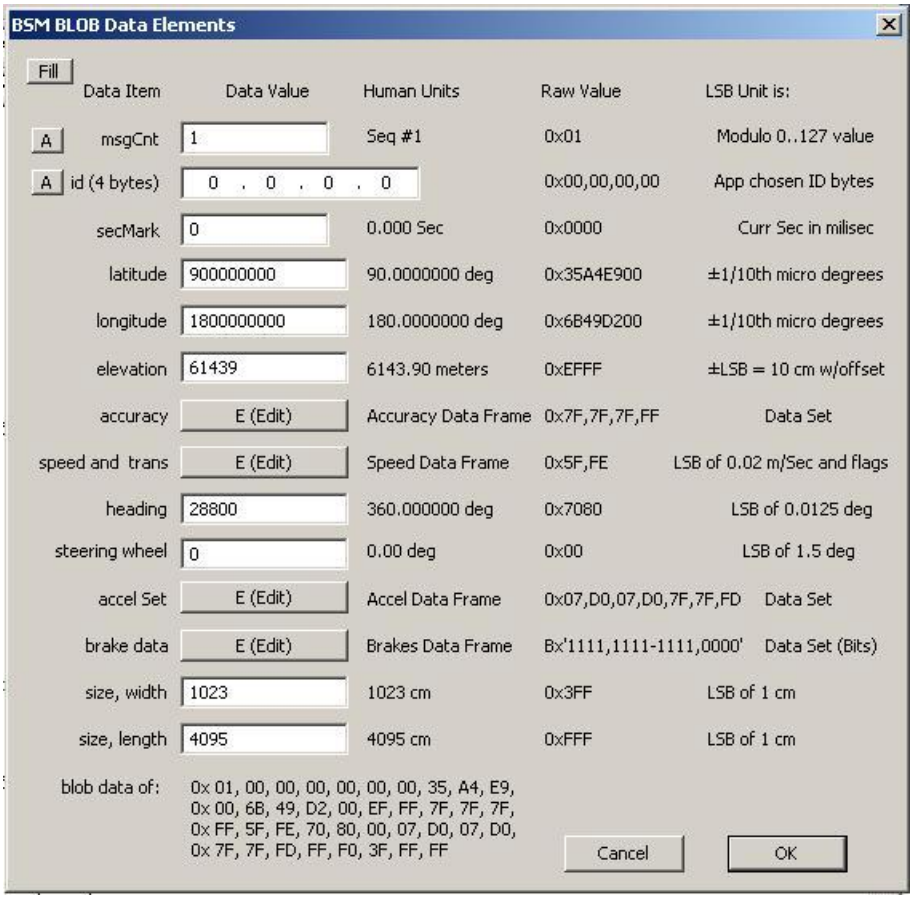

Figure 14 – Blob Content BSM Data Shown in the DSRC Tool

Using either the provided code example, or one of the suggested COTS tools, one can easily create additional messages with valid encoding in a matter in minutes. The output of these can be used to compare against messages created by one's own developed code to ensure conformance with the message set.

3.7 Relevance to ATIS or ITIS Message

Because the Basic Safety Message makes very little use of any ITIS codes and no use of ATIS, this subsection is limited for this application. The treatment of ITIS codes and ATIS messages is deferred until the next application is discussed (Roadside Warnings) which heavily uses the ITIS encoding system.

3.8 Interface Design Issues/Considerations

When building the decoder logic for receiving and processing the Basic Safety Message (or any of the DSRC messages) it is typical to hand the stream of bytes received from the listener socket directly over to the ASN library decoder. Then, if the message is decoded without error, the results (message content) are returned in the *.h file structures defined by the ASN.1 library and can be processed further. In messages which contain data blobs (such as found in the BSM) it is also typical to add a second level of decoding to extract the blob information and place it into a suitable structure. The resulting final structure is then sent or exchanged with the various applications that have a need to receive it, using some method of message dispatching or sharing. If a given application does not need all of the content of the message, some savings may be obtained by decoding the message in sequential parts, but few ASN.1 toolkits support this process. See Figure 15 in subsection 3.9 for a graphic of decoding where several different applications have differing final data needs from the same message.

It is possible (but not advisable) to construct this message by adding current data values into a pre-built *shell* of ASN encoding, adding or removing the values to the known locations. For example, the 32 bit value (4 bytes) of latitude is always found as the first four bytes of the blob and can simply be added there. The single byte value representing the `msgCnt` can be added directly to the seventh byte of the message. Such an approach is very brittle to any subsequent changes in the ASN, but it has its place in very low-end applications. It is of course required to create blobs with internal structural meanings. This style of development (the use of blobs and the direct extraction of value fields) is only reasonable to undertake when no optional elements are present in the preceding stream of elements, because any such elements cause a movement in the assigned byte locations, depending on what optional element(s) and data value sizes preceded that location. If the prior element values vary in length, this can quickly become very complex to track and it is better to use the ASN library to encode the message anew each time. Some ASN libraries provide calls to detect where the value field starts for a specific data element, and can therefore support this sort of behavior. This style of programming rarely saves any processing time over letting the library process the message, and is difficult to maintain over the project lifecycle.

3.9 Message Sharing Issues/Considerations

The Basic Safety Message, once received, is shared by many safety applications; perhaps as many as a dozen independent processes will want to use the same data (see the list outlined in subsection 3.2 above). Each of these applications has different needs from the message, and most do not need every data item present in the message.

As a result, there is a need for message sharing to some degree, and perhaps this will affect the ordering and processing steps in which message decoding is undertaken. The image below shows three such example applications (App A, B, C). In App C there is no need for additional decoding to

extract the blob data, so it can utilize the message after the ASN.1 library has decoded it. App B does require the additional blob data and therefore a second stage of decoding must be invoked to unpack and obtain that data. Finally, App A also uses the blob data but requires it to be formatted into another structure of some sort (perhaps it is reflecting incoming BSM positions from the receiving vehicle's current position and therefore must translate each message into a new coordinate system). These three applications are simply illustrative examples of what may be found in a real time system processing DSRC messages.

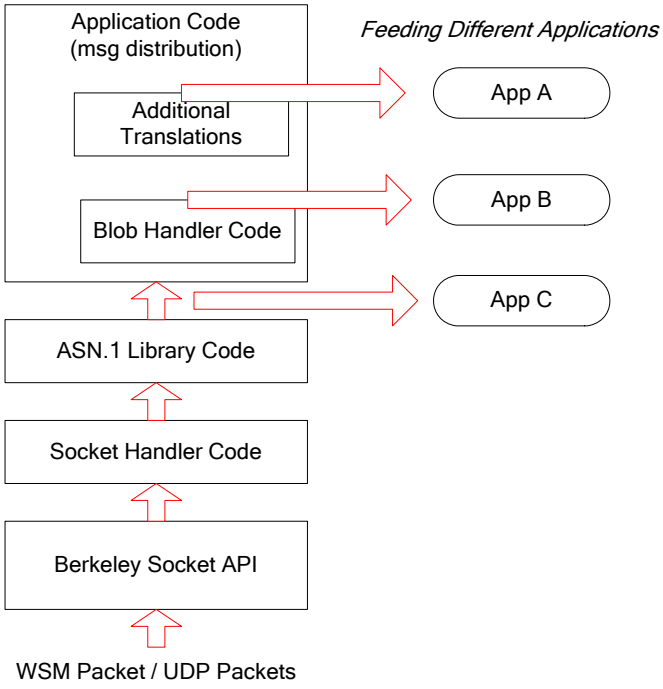

Figure 15 – An Example of Multiple Applications Using the Same Message

Skills Check

After a review of this section and some time downloading and playing with the supplied tools, you should now:

1. Be able to create a simple BSM using the supplied tools with your own chosen content.
2. Be able to decode a simple BSM in its ASN component parts using the supplied tools.
3. Be able to recognize the basic T-L-V groupings in a simple ASN message.
4. Be ready for additional details on more complex encodings (the next section).

4. Application 2
Immediate and Roadside Generated Warnings

4.1 Description

The `RoadSideAlert` message is one of several related messages found in the DSRC message set that can be used to relate local and regional traffic conditions using the ITIS codes. Quoting from the standard:

> *... [this] message is used to send alerts for nearby hazards to travelers. ... Typical example messages would be "bridge icing ahead" or "train coming" or "ambulances operating in the area." The full range of ITIS phrases are supported here, but those dealing with mobile hazards, construction zones, and roadside events are the ones most frequently expected to be found in use.*

The emphasis on *mobile hazards* is further reflected in the *MSG_EmergencyVehicleAlert* message which uses this same basic message but adds additional information about the type of vehicle and where the emergency vehicle is traveling (a position and motion vector) so that nearby OBUs can react accordingly. Other messages in the standard, such as *MSG_TravelerInformation,* (which extensively covers signage support) also use these codes. In addition, the ATIS event message, encoded in XML and available over the IP protocol of DSRC, also makes use of the same ITIS codes.

This message is typical of the widespread use of the ITIS codes found in the ITS messages to encode event information. It was chosen as a guide example to illustrate proper ITIS code handling in the ASN DER environment of the WSM. The ITIS codes are used in all of the major message sets (ATIS, TCIP, TMDD, IM) as the common industry-wide methodology to characterize, classify, and describe events; they currently consist of over 2000 defined codes. Any DSRC device receiving these messages needs to implement some form of ITIS decoder (a topic of discussion further in this section). The ATIS Users Guide (see the References subsection 1.5) contains extensive notes on the use and handling of ITIS codes in messages, typically with XML examples. The ATIS Users Guide is freely available from the SAE.

The `RoadSideAlert` message is a broadcast style message sent over the WSM protocol at short repetitive intervals during its duration. The term *roadside alert* is something of a misnomer: the messages themselves can be generated and sent not only from permanent and temporary roadside devices, but from other mobile vehicles' OBUs as well (when properly signed and authorized). In this way, public safety vehicles and response equipment can broadcast the critical situational information to travelers without recourse to any network infrastructure. This allows not only on-scene *ad hoc* broadcasts of information to travelers for incident management, but for responding vehicles to broadcast alerts and warnings while moving to and from the scene.

Vehicle based applications will typically receive all of these messages and use some form of common ITIS decoder routine to decide how to present the received information to the driver. This can vary with the model and features of the device and is outside the scope of concern here. In our examples, we simply display a textual form of the relevant ITIS codes. In our applications we simply recover and decode the message, comparing each to previously received messages to detect duplicates in content. Duplicates are expected because this style of message is routinely broadcast with static content at a repetitive rate. Often the information is static for long periods of time (as long as many months at a time for roadside construction events). A simple algorithm based on the Cyclic Redundancy Code (CRC) of the message payload is used to detect duplication; however, what to do in such a case is an application-level issue.

4.2 Relevance to the VII Architecture

The premise of VII and the resultant architecture is to provide services to enhance mobility, safety and commerce. The provision of traveler information to drivers is one of the principle requirements to enhance mobility. As illustrated in the BSM example applications in section 3., situational awareness requirements support multiple applications. In this case, one vehicle provides data to others, or a local RSU provides the data. In the case of emergency warning messages, vehicle specific data regarding its position and motion vector data is added to the basic event information provided by the ITIS codes in the message. For example, a safety service patrol may, while parked on the side of a freeway, send information concerning the fact that it is currently assisting a stranded motorist. Portable message signs upstream of work zones may provide information and warnings related to the work zone configuration.

Although this function does not require the network component of the architecture to operate, the central network functions are still needed to provide security verification and Certificate Revocation Lists (CRLs). Without such certificates and CRLs, the security and authentication aspects of the process will not work. The use of the VII network even intermittently allows such applications to be kept current and such mobile devices to update their authorizations and credentials.

4.3 Communication Parameters

Immediate and Roadside Generated Warnings

Minimum Required Communication Range	There are least three use cases here: • Construction Use, Incident Use, other *ad hoc* deployments • Established standing warnings (bridge ice type, other signage warnings) • Lower grade short range messages such as school zones, loading zones, etc. Ranges on each varies
Message Direction	Mostly RSU to OBU but some OBU to OBU cases when incident response equipment/vehicle is used (both a vehicle OBU or some form of mobile RSU equipment such as sign boards); all one-way broadcasts
Broadcast Interval	Nominally 10 per second, less for some standing warnings (1~3 per second); standard does not specify
Priority Assignment Criteria	May vary based on message content; committee debate and assignment of the correct range is still subject to change

4.4 How to Use the Message

The `RoadSideAlert` is one of several messages in DSRC that use ITIS and are grossly based on the traveler advisories found in the ATIS message set standard. Like ATIS, they serve to inform the traveling public of various events, and use the ITIS coding for that purpose whenever possible. Unlike the ATIS work, they do not allow extensive use of free text[33] to describe the event for

[33] The common ITIS pattern of mixing codes with free text to more fully describe the event can be used to express things in only free text (ignoring the use of ITIS codes except for the "key" phrase). This ability was placed into the ATIS standard to allow existing deployments that had only a free text system to create valid messages, but as a technical approach it is strongly discouraged for new designs.

bandwidth conservation reasons. Hence, the ITIS codes take on more significance in these messages. The language patterns of ITIS code use which are seen here are also found in other DSRC messages and the functionality developed to handle the codes can be reused.

It may be helpful to first review the highlights of the basic event message developed in the ATIS work before moving to the scaled down version allowed by DSRC. Recall that the event message of ATIS can be sent over DSRC, simply by using the XML encodings[34] defined in that standard and creating an IP style encoding. The event message allows a great many additional (and optional) data items, such as complex valid time sequences for reoccurring periodic events, or links to additional multimedia support. The event message also uses the *head* data structure to create relationships between such messages. For example, a secondary accident[35] can be linked back to the original accident, which in turn can be linked to the inclement weather that contributed to it. This type of linking is not supported[36] in the DSRC messages. Here is a simplified view of the event message, taken from the ATIS guide documents:

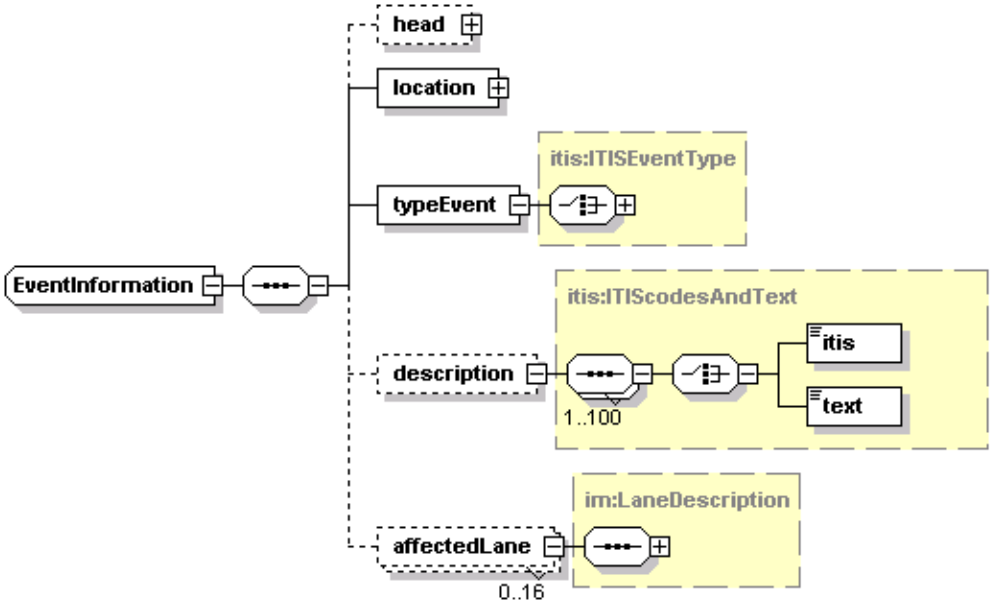

Figure 16 – ATIS Event Information Message (simplified)

This message is used to relate all sorts of event information. It begins with a location value for the event (a complex structure which supports all of the LRMS profiles defined), then a required ITIS Event Type (often called the "key phrase" as this single code is used to classify the event), and then adds additional details in the form of sequences of other ITIS codes interspersed with free text. As mentioned before, there are several dozen other optional elements which are not shown. The variation allowed makes the event message too long for use over the WSM protocol.

[34] XML encoded messages are never sent over the WSM protocol where this message is used.

[35] Secondary accident: An additional event that occurs after an initial event, typically near the same location, and due to fallout from the original event. A rear-end impact accident in the other lanes across from the first accident due to onlooker slowdown is the typical example used. Some agencies use this term to refer to active events in their Computer Aided Dispatch (CAD) for post event infrastructure reconstruction tasks.

[36] There is however an ID value which can be used to link back into one or more ATIS messages.

By contrast, the DSRC Roadside Alert is shown below. Note that the location (here called *position,* an instance of the *FullPositionVector* and optional[37]) is again present. The location data supported is much more limited in this context (and is discussed further in a moment). We again have a single ITIS code (*typeEvent*) to represent the general classification of the event, and we again have a sequence of additional ITIS codes to provide further details (*description*). (ITIS code use is discussed further in a later subsection.) Part of the design intention is to be able to easily create Roadside Alert messages from the data already encoded in ATIS event messages.

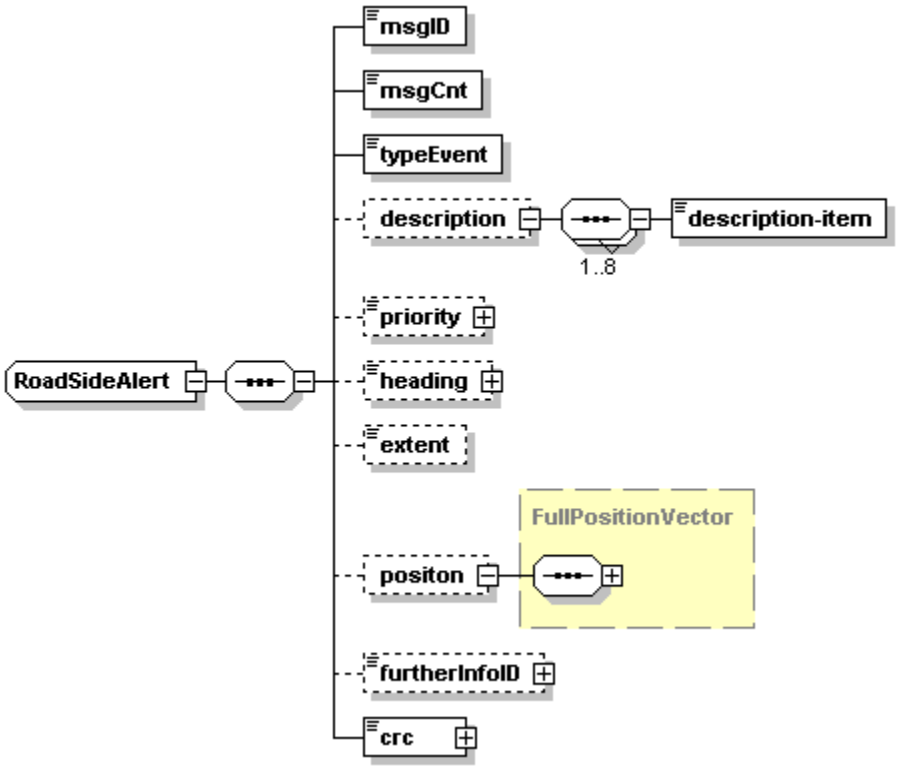

Figure 17 – DSRC Roadside Alert Message Structure

Note the message begins with the *msgID* field used to denote that this is an RSA message type (the value eleven is assigned to RSA), and the *msgCnt* field is used to provide a running count of the messages issued (used to detect when a message iteration was lost). Each of these are simple integer values and are assigned values in the ASN library routines in the normal way. Lets consider a few of the other simple elements found here.

The optional *priority* data element allows sending a relative level of importance for the message. The optional *heading* data element allows sending a general direction of travel to which this message should apply and is considered further in a moment. The optional *extent* data element allows sending an indication of the duration or length over the roadway for which this message is valid. Because of the optional content, the basic message can be very simple, consisting of from one to nine ITIS codes. If all the optional elements are not used, this message can consist of only the *typeEvent* data element and the various message framework items (Id, Cnt and crc).

[37] These images come from the XML representation of the messages. You can find this image in the online documentation of the DSRC *schema http://serv1.itsware.net/itsschemas/DSRC/DSRC-03-00-36/docs/#element_roadSideAlert_Link07058258*. See also, the ATIS Users Guide documentation for additional ATIS details, or consult that standard, *http://www.itsware.net/itsschemas/ATIS%20Guide/AGuidetotheATISGuides/*

When expressed in its ASN format, this message is as defined below:

```
RoadSideAlert ::= SEQUENCE {
   msgID          DSRCmsgID,
                  -- the message type.
   msgCnt         MsgCount,
   typeEvent      ITIS.ITIScodes,
                  -- a category and an item from that category
                  -- all ITS stds use the same types here
                  -- to explain the type of  the
                  -- alert / danger / hazard involved
                  -- two bytes in length
   description    SEQUENCE (SIZE(1..8)) OF ITIS.ITIScodes OPTIONAL,
                  -- up to eight ITIS code entries to further
                  -- describe the event, give advice, or any
                  -- other ITIS codes
                  -- up to 16 bytes in length
   priority       Priority OPTIONAL,
                  -- the urgency of this message, a relative
                  -- degree of merit compared with other
                  -- similar messages for this type (not other
                  -- message being sent by the device), nor a
                  -- priority of display urgency
                  -- one byte in length
   heading        HeadingSlice  OPTIONAL,
                  -- Applicable headings/direction
   extent         Extent OPTIONAL,
                  -- the spatial distance over which this
                  -- message applies and should be presented
                  -- to the driver
                  -- one byte in length
   position       FullPositionVector OPTIONAL,
                  -- a compact summary of the position,
                  -- heading, rate of speed, etc of the
                  -- event in question. Including stationary
                  -- and wide area events.
   furtherInfoID  FurtherInfoID OPTIONAL,
                  -- a link to any other incident
                  -- information data that may be available
                  -- in the normal ATIS incident description
                  -- or other messages
                  -- 1~2 bytes in length
   crc            MsgCRC
   }
```

A critical conceptual difference between the use of this message and the event one is the treatment of location. Because DSRC messages are sent over a relatively short and localized area, there is the presumption that any valid message an OBU receives applies to the nearby area in which it is currently traveling.

We now examine the support for location a bit further. The `FullPositionVector` data frame is used to relate geographical information about the message and is defined as:

```
FullPositionVector ::= SEQUENCE {
   utcTime             DDateTime OPTIONAL,    -- time with mSec precision
   long                Longitude,             -- 1/10th micro degree
   lat                 Latitude,              -- 1/10th micro degree
   elevation           Elevation OPTIONAL,    -- 3 bytes, 0.1 m
   heading             Heading OPTIONAL,
   speed               TransmissionAndSpeed OPTIONAL,
   posAccuracy         PositionalAccuracy OPTIONAL,
   timeConfidence      TimeConfidence OPTIONAL,
   posConfidence       PositionConfidenceSet OPTIONAL,
   speedConfidence     SpeedandHeadingandThrottleConfidence OPTIONAL,
   ... -- # LOCAL_CONTENT
   }
```

Expressed in its XML graphic format, this is:

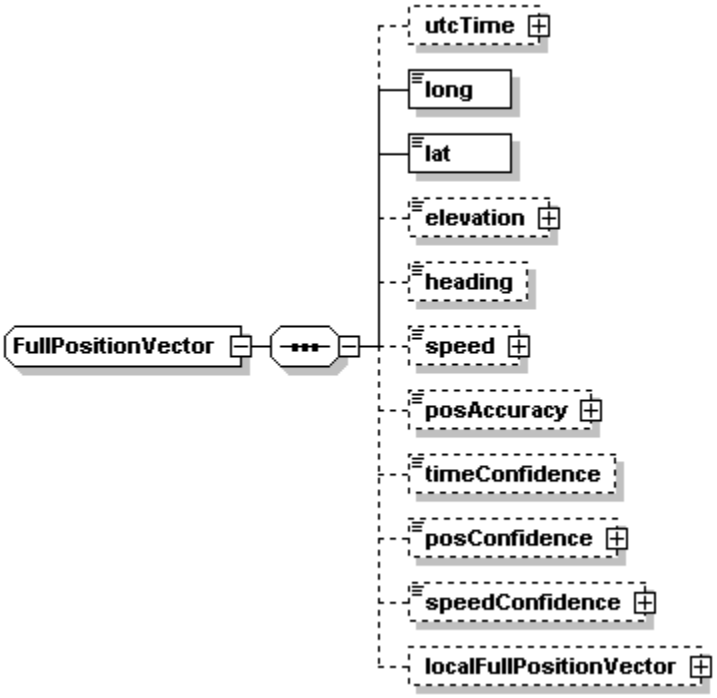

Figure 18 – Full Position Vector Data Structure

Location in this context is used to relate the (lat-long) location for the event or of the sender (in the case of moving objects such as pubic response vehicles). This entire data frame is marked `OPTIONAL` because there is the presumption that if you are within range of receiving the message, it in fact applies to your location. Often the elevation data element is not used. When it is used it may apply to road conditions at elevation levels about the stated value.[38] There is a further presumption that some roadside equipment broadcasting this type of message (rudimentary mobile message boards) may not know their geographical position, or may not get a position assigned during the initial emergency response period. The `OPTIONAL` elements for *utcTime* and for the various *confidence* values are typically not used when the *FullPositionVector* (FPV) is used in the RSA message.

A second usage of location concepts can be seen in the form of the data element (outside of the FPV[39]) called *heading*, which is an instance of the *HeadingSlice* data element. This is used when a message should be applied only to traffic or travelers moving in a specified direction of travel (an accident report might apply only to westbound traffic). Note that FPV frame also allows a field for the heading of the sender; this is different than the data element for sending the heading to which the message should apply. This *heading*, and the *speed* data elements of the FPV, are used to detect if a moving message sender is on a motion heading which will close with the receiver unit.[40]

The *heading* data element is a copy of the *HeadingSlice* element. It is composed of a 16 bit field (2 bytes) representing 22½ degree slices of the unit's circle to which the receiver can match the vehicle's general sense of direction. As shown in the image below, each nibble represents one

[38] Often used for weather reports and dangerous travel conditions, messages such as "chains required above 4000 ft".

[39] Do not confuse the *heading* that is inside the Full Position Vector (FPV) with this heading that is part of the RSA message; they are different element types even though they share a similar instance name.

[40] Typically used with messages that relate that emergency responder vehicles are operating in the area.

quarter of the unit circle. Traffic is presumed to travel from the center to the outside. So, for example, traffic heading due eastbound would be indicated by setting two bits with the value `0x0018`. Any bits can be set; all bits being set would indicate a message that applied to all directions of travel (all receivers in that area). A default setting of all zeroes here may indicate that no directionality would apply, or it may indicate that the direction is simply unknown. Weather reports are a good example of such a case.

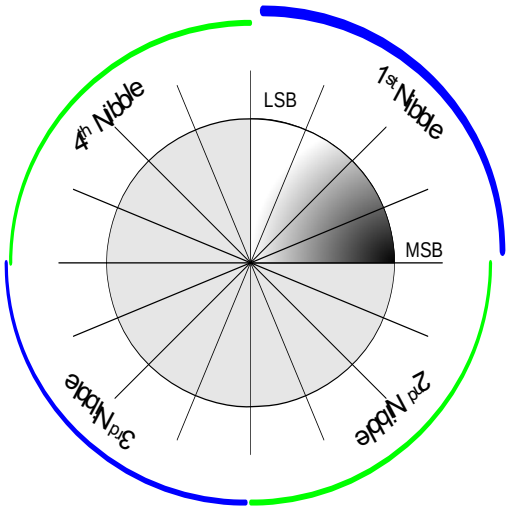

Figure 19 – Heading Slice Data Structure

Finally, the *extent* data element is used to optionally provide gross sense of persistence or duration for the application of the message. Some messages apply only when received, while others may apply and be valid over a length of roadway.[41] This enumeration allows a basic persistence to be denoted.

```
Extent ::= ENUMERATED {
    useInstantlyOnly   (0),
    useFor3meters      (1),
    useFor10meters     (2),
    useFor50meters     (3),
    useFor100meters    (4),
    useFor500meters    (5),
    useFor1000meters   (6),
    useFor5000meters   (7),
    useFor10000meters  (8),
    useFor50000meters  (9),
    useFor100000meters (10),
    forever            (127)  -- very wide area
    }
    -- encode as a single byte
```

When an OBU receives the `RoadSideAlert` message it must determine if it has received this same message (same content) in the recent past. If so, it may be able to discard the message. Consider a school zone speed warning. The vehicle is likely to receive a number of these (static) messages while passing through the zone. The on-board logic would (may) alert on the first reception, but is not likely to alert on each subsequent message. The rules controlling this are complex and will vary

[41] In other cases, there may be no further DSRC infrastructure on the roadway ahead, and this message can be used to send general warnings about traveler conditions along that stretch of road. For example, going into mountainous areas where the RSU infrastructure may be nonexistent, it may be useful to issue warnings and cautions about snowfall, falling rocks, chains required, etc., as conditions warrant.

with each OEM developer. The key item is still needed: an effective way to detect duplicate messages, and the *MsgCRC* data element fulfills this need. The contents of this data element (always the last two bytes in the message payload) are computed by running a CRC algorithm over the message payload directly to determine its value. It is computed such that the CRC computed by the receiver device over the entire message will always be zero if the contents have arrived intact. In this way the CRC can be used both as a hash and as a checksum.

4.5 Representative Encoding of the Message

It is very important at this point to recall again that the ASN.1 of the standard describes the structure and ordering of the message *over the air* between systems. It does not describe how the same message is kept in the memory of each system at the sending or receiving end.[42] In fact this typically differs with the conventions used by each ASN.1 tool vendor. A precise one to one mapping is not possible for a number of reasons, as is illustrated in greater detail in section 6. This is most often seen in the way complex nested structures are treated (using pointers).

The various elements of the `RoadSideAlert` message provide a number of typical ASN types and values, and the process of setting these in code are briefly examined in this subsection. Readers with less interest in how to code such things can skip this and proceed to a review of what the resulting messages look like in their binary encoded forms. It is also useful for this subsection to refer to the source code provided with the application tool to see more of each type in use.

As a short summary of the ASN types that must be dealt with for the RSA components we have:

Table 6 – Types of ASN Objects Used in the RSA Message

Object	Instances
Simple enumerations	DSRCmsgID, ITIScodes
Simple integer values	MsgCount, and others
Simple (long) integer values	FullPositionVector, including Longitude, Latitude
Bit fields sets	HeadingSlice
List objects	Sets of ITIS.ITIScodes in a list
Complex data frames	FullPositionVector
Octet Strings	FurtherInfoID
The CRC element	MsgCRC

The `RoadSideAlert` message in itself is fairly flat and simple, so apart from the nesting of dialogs and routines needed to handle the optional *FullPositionVector* data frame, the building of message in code is straightforward. We begin again with the representation in memory produced by the ASN library tools when given the ASN of the standard. In the file *RoadSideAlert.h* we can find the data structure `RoadSideAlert_t` defined as follows:

```
/* RoadSideAlert */
typedef struct RoadSideAlert {
    DSRCmsgID_t      msgID;
    MsgCount_t       msgCnt;
    ITIScodes_t      typeEvent;
    struct description {
        A_SEQUENCE_OF(ITIScodes_t) list;
```

[42] These may differ depending on how each side has implemented the message and what choices of ASN.1 tools and libraries and options are selected to be used. Interoperability in this context is defined by the message exchange, not how the message is kept, processed, or used.

```
        /* Context for parsing across buffer boundaries */
        asn_struct_ctx_t _asn_ctx;
    } *description;
    Priority_t                *priority             /* OPTIONAL */;
    HeadingSlice_t            *heading              /* OPTIONAL */;
    Extent_t                  *extent               /* OPTIONAL */;
    struct FullPositionVector  *positon             /* OPTIONAL */;
    FurtherInfoID_t           *furtherInfoID        /* OPTIONAL */;
    MsgCRC_t     crc;
    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _asn_ctx;

} RoadSideAlert_t;
```

Like the BSM memory representation used in section 3., this listing immediately shows that some items (`description, priority, heading, extent, positon,` and `furtherInfoID`) are referred to by pointers while others (`msgID, msgCnt, typeEvent,` and `crc`) are not. Again, the use of the pointer is this tool's way of indicating that the content specified is `OPTIONAL` and if the pointier is null no such content is present.

Note also the strange incantation used to hold the collection of ITIS codes that make up the description:

```
struct description {
    A_SEQUENCE_OF(ITIScodes_t) list;
    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _asn_ctx;
} *description;
```

This is the tool's way of making a list or collection and will be dealt with in turn. Note that it too uses a pointer because the description list is itself `OPTIONAL` to use in the message.

In order to build one of these messages programmatically (that is, in a code environment) one begins with instantiating a copy of the message type itself, with code similar to that shown before.

```
theMsg = (RoadSideAlert_t *)calloc(1, sizeof *theMsg);
```

The above occurs as part of the initial dialog routines to start up the dialog that displayed this message type. To this, various data elements can be added, always keeping in mind the prior warning that when a pointer is used, the responsibility to allocate space for what is pointed too (and to destroy it later) falls on the programmer.

Lets us start with the `msgID`, which is simply an enumerated value assigned to the data element

```
theMsg->msgID = DSRCmsgID_roadSideAlert;
```

To this we can also add a *msgCnt* value, which for our initial message can be the value of zero.[43]

```
theMsg->msgCnt = m_msgCnt;  // msg Cnt
```

We can now add the selected "key" ITIS code that characterizes this message. Recall from the data on proper ITIS use[44] that this code is used to characterize the general type or category of message that this is, so typically the choice is made to use the most significant code in the overall description. Any two-part code concepts, such as *fire canceled* would then use "fire" rather than "cancel" for the code.

[43] Recall from the definition given in the standard that this value increments up to 127 then again returns to zero, forming a module base 127 count value in the stream of messages.

[44] See numerous other materials on the proper ITIS code selection including the ATIS guides.

The coding itself is simply another enumerated assignment, therefore similar to how *msgID* was treated. But in this case we have not imported all of the ITIS codes into our programming environment as enumerated string constants,[45] so we simply assign the numeric value itself.

```
theMsg->typeEvent = (short)theApp.ITIS.msgSetValues[0]; // event type
```

In the above we are setting the value based on the first entry in an array from an application global called `theApp.ITIS.msgSetValues` but the following would do the same (if we wanted to assign the ITIS code value 513 for *accident*):

```
theMsg->typeEvent = (short)513 // event type = accident
```

Now we come to an interesting case, the use of the "list" by the tool to build up the contents of the `SEQUENCE OF` statement from the ASN side.

We again use the global structure we have created to hold ITIS codes (`theApp.ITIS.msgSetValues`) and we need to loop over its contents and assign them to the *description* data frame. However, we must first allocate space for the *description* itself, and then allocate space for each element we assign. Finally, we attach this new structure to the message instance itself (making the pointer for this *description* in the message now point to it rather than be null). The code looks like this:

```
description* desc = (description* )calloc(1, sizeof(description)); //create
for (i=1; i<m_itisItems; i++)
    // for each code we have in the global array
    {
      code[i] = (ITIScodes_t* )calloc(1, sizeof(ITIScodes_t)); // create
      code[i] = (ITIScodes_t* ) &theApp.ITIS.msgSetValues[i];  // assign
      asn_sequence_add(desc, code[i]);                         // attach
    }

    theMsg->description = (struct RoadSideAlert::description *)desc;
       // attach to main msg
```

Notice the call to `asn_sequence_add()` which is a helper function provided in the ASN library file *asn_SEQUENCE_OF.h* to manage the collection of items in a `SEQUENCE OF` list object. This function manages the list pointers and adds the passed object (`code[i]` of type `desc`) to it.

The *priority* is a case where a pointer is involved. Therefore we need to have allocated space to keep the *priority* value in at some time before its use. In this code we have taken the approach to allocate the *priority* once during the life of the dialog and then simply assign to it when needed; the interesting code appears over the next three code fragments. Thus the critical code to build up the element appears as (part one of three code fragments to follow):

```
//if IDC_CHECK_priority is set...
theCkBox = (CButton*) GetDlgItem (IDC_CHECK_priority);
if (theCkBox->GetCheck() == BST_CHECKED)
    theMsg->priority = &thePriority; // point to it
    // NOTE - IMPORTANT
    // Note key point to learn:  When we change this pointer from NULL,
    // the ASN lib code then knows that the optional item is present!
    theMsg->priority->buf[0] = m_priority;
    // fill it in with member data
else
{   theMsg->priority = 0;
    // pointer is set to null (may be pointing to memory
    // from prior data set)
}
```

[45] A useful and suggested step but not one done here as it causes the code size to grow considerably. Importing the codes from the adopted ASN source listing is easily done by those who wish to add it.

If there was no desire to use this optional element, the pointer would simply remain set to null. The outer *if-then-else* clause is used with a visual "checkbox" control to determine if the *priority* value will be added to the message; this is a pattern we commonly reuse in the example code. The global value we have used here (`thePriority`) was allocated and created as follows during the start up phase for the dialog that handles RSA messages (part two) with:

```
thePriority.buf = (uint8_t *) calloc (1, 1);  // one byte
assert (thePriority.buf);
thePriority.size = 1;
```

And this was set to a valid value beforehand (where the combo user interface dealt with the drop down combo box that holds a list of valid *priority* values) as such (part three):

```
CComboBox * pControl = (CComboBox*) GetDlgItem (IDC_COMBOpriority);
m_priority = (BYTE) pControl->GetItemData(pControl->GetCurSel());
theApp.log.AddRtxtReport(enInformative, "Priority value changed.", false);
```

This divided style was adopted to make the crucial parts of the code easier to read by those not familiar with the MS Windows environment. In general, each variable is set using Windows visual interfaces (such as part three), but the action of building up the message using those values is all done in a routine called *BuildMsg()* which is of a more direct traditional C style (parts one and two above). For those not interested in the issues of MS Windows programming,[46] the first two code fragments should be sufficient.

In the above example, the control used for the visual interface is simple: a combo list was preloaded with valid values from which to chose. For more complex structures, especially data frames, we typically have added a new dialog to the user interface to break the internal components down to simple items for user control and input. An exception to this is the `HeadingSlice_t` *or heading* data element, which, although just a 2 byte octet value, is worthy of a dialog of its own.

The methodology used for this simple dialog is also used for more complex ones. The data in question is copied into the dialog as either an octet stream (as in this case) or as a C structure with all the component elements. Inside the dialog code itself, this is disassembled into the component parts and displayed. When the dialog is dismissed, it is entirely rebuilt and passed back to the caller. This is another design pattern that is commonly reused in the application code.

Considering how the heading is initialized, we use a simple copy operation to send the two bytes of data to the dialog class. While only two bytes are used here, in other cases more are found (for example the *BSMblob* requires 38 such bytes to be exchanged).

```
// load in current value set to dialog
theHeadingDlg.m_heading[0] = m_heading[0];
theHeadingDlg.m_heading[1] = m_heading[1];
// invoke dialog now, processing it further if we get an ok return
```

And when the dialog is dismissed (with an OK button press) we have the below code to extract and use the returned values in the message[47]:

```
// get the new data back from the dialog
// note that validity testing logic was done inside the dialog box
m_heading[0] = theHeadingDlg.m_heading[0];
m_heading[1] = theHeadingDlg.m_heading[1];
theMsg->heading = &theHeading;              // assign it
theMsg->heading->buf[0] = m_heading[0];     // load member var
theMsg->heading->buf[1] = m_heading[1];
```

[46] Section 8. dwells on these design patterns to a larger degree and would be useful to the reader that wants to have a better understanding of the variable naming used and the conventions between the visual control variables, the allocated buffer spaces to hold them, and the actual message in memory.

[47] The logic to test if the dialog was canceled or dismissed with an OK button and the logic to check if the optional element is to be assigned are not shown for clarity. This complete code can be found in the *RSA.cpp* file in the function that handles the heading slice dialog, `OnBnClickedButtonHeadslice()`.

The assignment of the original buffer space, as before, was handled in the dialog start up phase. Note again the use of variables named with the pattern *m_XXX* for the visual control value and *theXXX* for the final memory representation of the data concept, another design pattern which reoccurs in the application code.[48] We see this again when we deal with the *FullPositionVector* in a moment .

The next item in the message is the *extent* data element. The three code sections to *assign it*, *allocate it*, and let the user *manipulate it* all follow the same pattern just shown for the *priority* element and are shown below.

```
// Assign it to message
if (theCkBox->GetCheck() == BST_CHECKED)
{ // fill it with member var
   theExtent = m_extent;          // from member var to struc var
   theMsg->extent = &theExtent;   // (make not null)
}
else
{
   theMsg->extent = 0;
}

//Allocate it (in dialog init)
Extent_t            theExtent;

// Manipulate it
CComboBox * pControl = (CComboBox*) GetDlgItem (IDC_COMBOextent);
m_extent = (BYTE) pControl->GetItemData(pControl->GetCurSel());
theApp.log.AddRtxtReport(enInformative, "Extent value changed.", false);
```

Now we can address the *FullPositionVector* which is a complex data frame full of other smaller elements to be set. The routine described here is reused in the code for all other instances of the FPV and works the same there. In this design pattern we initially exchange a stream of structured elements (the `FullPositionVector_t` structure in file `FullPositionVector.h`) over to the dialog and return it when done, by way of a passed pointer. Because this is a defined structure, the code involved is simply:

```
// create m_pPos and copy to it from existing data (not shown)
theMsg->positon = m_pPos;               // make msg point to it
theFullPosDlg.m_Pos = theMsg->positon;  // assign the dialog ptr
```

Once “inside” the dialog the individual elements are allocated, set, and manipulated just like any of the other element types. For example, *latitude* and *longitude* values (which are required and not optional in this structure) are assigned directly:

```
// do the lat-long first (these are not optional)
aFPV->lat = m_latitude;
aFPV->Long = m_longitude;
```

while *elevation* (which is optional) is assigned with the normal checkbox logic (not shown) and a newly created buffer to hold it, as in:

```
aFPV->elevation = new Elevation_t;
aFPV->elevation->buf = new BYTE[2];
aFPV->elevation->size = 2;
aFPV->elevation->buf[0] = (BYTE)((m_elevation & 0xFF00) >> 8);
aFPV->elevation->buf[1] = (BYTE)((m_elevation & 0x00FF) );
```

All the other elements of the FPV are handled in similar ways.

[48] It perhaps should be pointed out in passing that the creation of two separate data items to hold the same data concept of a message is a choice made because this is an interactive visual tool and not an embedded application, where such a duplication in memory would be wasteful and not used.

Observe that *UTCTime* is yet another complex data frame nested inside the complex data frame of the FPV. This causes yet another dialog to be created to deal with its contents, which are in turn returned to the FPV in the same way when completed.

In order to complete filling out the RSA memory footprint, we still need to fill in two further elements, that of *FurtherInfo* and the *CRC* elements. The *FurtherInfo* element is simply a 2 byte long octet string and is handled with the same assignment methods as other octets.

```
theMsg->furtherInfoID = &theInfo; // assign it
theMsg->furtherInfoID->buf[0] = (BYTE)((m_furtherInfo & 0x0000FF00) >> 8);
theMsg->furtherInfoID->buf[1] = (BYTE)((m_furtherInfo & 0x000000FF) );
```

The CRC element is somewhat more complex because it requires two steps in order to compute its value. First, one must run the `CRC-CCITT` algorithm over the entire message without the CRC being present, and then one must use the resulting values to create a CRC and stuff it into the rest of the message (the last two bytes). In the applicator code we automatically recompute the CRC any time any of the other message components have changed. A short timer is used to detect that no further input has occurred and when that times out (in tens of milliseconds) the CRC is updated. The actual CRC routines are found in the utility code file (*Cutil.cpp*) in the function `DoCRC()` and is called by:

```
// theBytes holds a valid serialized message sans CRC values
m_crc = (unsigned short)(theApp.util.DoCRC(theBytes, i-2));
m_crcComputed = m_crc;
byteArray[msgLength -1]  = HIBYTE(m_crc);
byteArray[msgLength -2]  = LOBYTE(m_crc);
// note that the message payload is complete at this point
```

At this point we now have a populated memory version of the RSA message. We can encode this into the payload (the ASN DER encoding) with a call similar to that used by the BSM example such as:

```
i = (int) DSRC_serializer(theMsg, theBytes); // into DER here
computeCRC(theBytes)
theBytes[i-1]  = HIBYTE(m_crc);
theBytes[i-2]  = LOBYTE(m_crc);
// tell the user about it with a nice printout to log
text.Format( "The Roadside Alert Message encoded into %i bytes of BER-DER:
```

Note that the last thing we did is add the final two bytes for the CRC value after the serialization process has been completed (with fake byte values present for the CRC as placeholders[49]) in order to compute the right values for use.

The above has provided a rather lengthy exploration of the various fragments of code used to build up this message type. Exploring the accompanying source code provided will allow understating how these routines fit into a richer programming application. In the next subsection we examine what each of the these same message elements looks like over the air in their T-L-V forms.

4.6 Examples of Well-Formed Messages

The simplest `RoadSideAlert` message is just the ITIS code “accident” (value 513) with no additional `OPTIONAL` data; this encodes into sixteen payload bytes as follows:

```
0x 30 0E 80 01 0B 81 01 00 82 02 02 01 89 02 7A AD
```

[49] If you look closely at the code in the file *RSA.cpp* you will observe that when the composite message is calculated, it is always processed twice, the first time to build up the message with a placeholder CRC and the second time to use the computed CRC value directly. A simple real time timer event is used to control this, and it serves to give the visual illusion that all the items are updated at once.

Like all DSRC messages, it begins with `0x30` (for sequence) followed by the remaining message length (in this case 15 more bytes). The first red box indicates the T-L-V for the *msgID* (its type and therefore how the rest is encoded). As described previously in section 3. and in section 6. on ASN tagging rules, this is the first tag inside the sequence (hence `0x80`), it is one byte long (as are all *msgID* values) and the value is `0x0B` which in base ten is the value 11, the enumeration value assigned to the *roadSideAlert* message type.

The next red box is the *msgCnt* value; it is the second tag (hence `0x81`), is a one byte long value (hence `0x01`) and that value is zero (the first valid number in the sequence of values that will change as the message is reissued, starting from zero and growing to a value of127 before zero again).

The next red box indicates the T-L-V for the *typeEvent* data element. As described previously in section 3. and in section 6. on ASN tagging rules, this is the third tag inside the sequence (hence `0x82`), it is two bytes long (as are all ITIS codes), and the ITIS value is `0x0201` which in base ten is the value 513.

The next red box is the *crc* value; it is the tenth tag in the message (hence `0x89`), is a two byte long value (hence `0x02`), and is set to `0x7AAD` as the CRC value. The remainder of the message, the six optional items which could appear between tags `0x82` and `0x89`, are all optional and not present in this message instance.

If you were to look up the value 513 in the ASN or XML of the adopted ITIS standard[50] you will find lines like:

In the ASN:

In the `Accidents and Incidents` subcategory we have:

```
accident(513),      -- Use when no further data is
                    -- available regarding involved
                    -- vehicle type
```

In the XML:

In the simple type:

`<xs:simpleType name="`**AccidentsAndIncidents**`">` we have:

`<xs:enumeration value="`**accident**`" id="`**_513**`" />`

These two entries are considered as equivalent[51]. Note that like all ITIS values, the value of 513 is composed of 2 bytes (02 and 01). And the upper byte (in this case =2) indicates that this particular enumerated value comes from group 2, which is called the `Accidents and Incidents` subcategory. All ITIS codes fall into one of over 50 gross classification categories; this is covered in greater detail in the ATIS guide material. For the moment, it suffices to say that complex filtering and data sorting systems can be created using these categories. In ITIS, the subcategory of zero (an upper byte of zero) is reserved and is historically used when a number or digit value needs to be expressed. Within each subcategory, the range of 128 to 255 is typically reserved for locally added phrases. Local deployments can also create entire new subcategories in the second byte range above 128. Because of the need for national interoperability in the ITIS codes for DSRC, the local content feature is not used in DSRC but may be found in other deployments using ITIS codes.

[50] This adopted ITIS standard can be purchased from the SAE; see the References subsection. Portions of it can also be downloaded in an electronic form.
As an ASN module at: http://www.itsware.net/itsschemas/ITIS/ITIS-04-00-03/ITIS_04_00_03_Source.ASN.txt
As an XSD schema at: http://www.itsware.net/itsschemas/ITIS/ITIS-04-00-03/ITIS-Adopted-04-00-03.xsd
and its local content at: http://www.itsware.net/itsschemas/ITIS/ITIS-04-00-03/ITIS-Local-04-00-03.xsd

[51] Strings between the two forms vary slightly for readability. For example, the dashes used in ASN are dropped in the XML forms. Nonetheless, the precise forms given in the standard are normative.

In describing the above message, we say that it is an `accident` type from the category of `Accidents and Incidents`. Additional event details may be added, and these all come from various subcategories as well, but it is only the very first code that provides the classification of the category.

When additional details are present, up to eight more codes may be added. These details can come from any of the ~50 existing categories and can be used to build up very complex events. Instructions to drivers can also be given using this method. As a more complex example, consider the following sequence of codes:

```
<itis> accident </itis>          <!—value 513   or 0x0201 -->
<itis> right lane </itis>        <!—value 8196 or 0x2004 -->
<itis> blocked ahead </itis>     <!—value 776   or 0x0308 -->
<itis> short delays </itis>      <!—value 1538 or 0x0602 -->
```

This comes from the first XML example of an event message[52] in the ATIS guide; it is an excerpt of four codes shown in their XML expression form. When encoded into the DSRC `RoadSideAlert` message, this becomes:

```
0x 30 1C 80 01 0B 81 01 00 82 02 02 01
   A3 0C 02 02 20 04 02 02 03 08 02 02 06 02
   89 02 34 AD
```

Note that adding the three new ITIS codes has resulted in the fourteen additional bytes added to the message, now 30 bytes long. The new content is shown on the second line in the above and the ITIS codes are underlined. The three new codes are enclosed in an outer tag for the *description* element (`0xA3`, the fourth tag) and a length (`0x0C`). These are not enclosed in a red box, but all 14 bytes are part of the T-L-V of the fourth tag. Note that the three inner items are tagged with `0x02` to indicate the universal integer type. Displaying this same message in an ASN viewer tool we have:

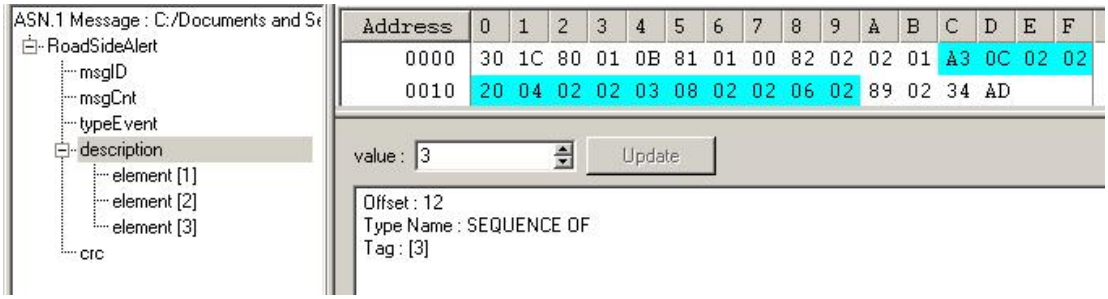

Figure 20 – Roadside Alert Message #1 Decoded

If we were to add some location data points to this message (again taking the additional data from the XML example mentioned above) we would have the following data elements to encode (shown in table form):

[52] For those curious regarding the equivalent XML encoding in the event message format, see the guide example fragments at: *http://www.itsware.net/itsschemas/ATIS%20Guide/ATIS-Events/examples/GuideEvent_001.xml*

Table 7 – Roadside Alert Message with Space Vector Data

Item	Human Value	Machine Value / Units
Lat	34.0833330	0x14-50-B4-32 1/10th micro degrees[53]
Long	-117.8666660	0xB9-BE-F9-5C 1/10th micro degrees
Vertical	skip	
Speed	0 (stationary, so skip)	
Heading	0 (skip)	
Key Phrase	"accident"	0x0201 (513)
Other Phrases	"right lane"	0x2004 (8196)
	"blocked ahead"	0x0308 (776)
	"short delays"	0x0602 (1538)
Heading Applied	Presume northbound traffic only	Two bits set, 0x8001
Extent	For 100 meters	0x04

This encodes as a message of 51 payload bytes as shown in the image below. Note that the new elements (added to the end of the prior message just before the CRC) consists of three T-L-V objects at the top level. The first (0x85,01,04) represents the *heading* data item, the next (consisting of 0x86,01,04) represents the *extent* object. The position data (starting with 0xA7 and highlighted) represents the *FPV* data item and its two inner content items (lat and long):

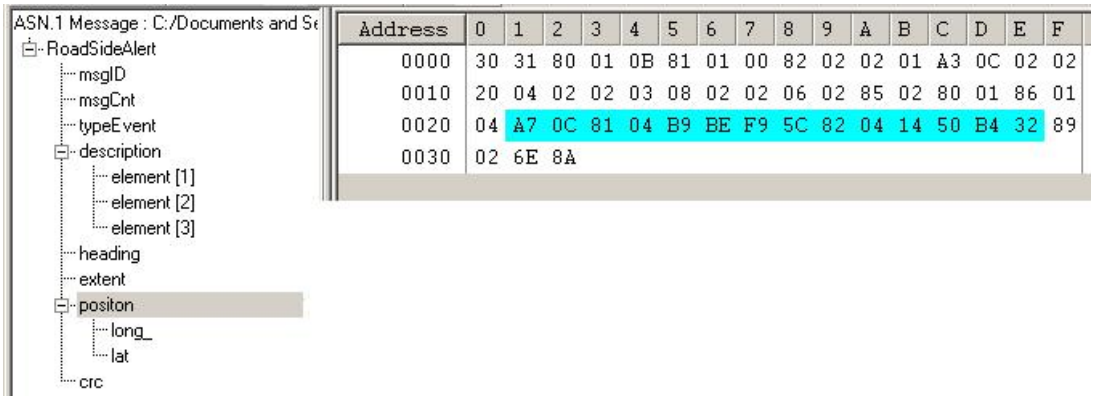

Figure 21 – Roadside Alert Message #2 Decoded

This instance of the *FPV-position* consists of two nested objects (*lat*, and *long_*), in all 12 bytes in length (0x0C). Note that the term "long" has been mangled to become "long_" because that word is a reserved keyword in many programming languages. The two inner objects are tagged in the normal T-L-V way (0x81 and 0x82) which should be familiar to the reader by this point. If not, a review of section 6., which deals with ASN tagging rules, may be helpful at this point.

Let us now consider a more complex example. In the ATIS guide is event example #3 dealing with a local disruption in traffic.[54] We also use the linking feature of the DSRC message to link back to the event message, which may be available on a service channel of the local DSRC system where bandwidth management is less of a concern.[55]

[53] We will use these location values for the remaining examples to aid the reader in decoding the bytes.

[54] You can see the complete event XML of this example at:
http://www.itsware.net/itsschemas/ATIS%20Guide/ATIS-Events/examples/GuideEvent_003.xml)

[55] The numbering and management of these link values is beyond the scope of this document, but of course is needed for the two messages (which may be generated, transmitted, and maintained by different agencies), to be coordinated.

If we again add some location data points to this message (taking the additional data from the XML example mentioned above) we get the following data elements (shown in table form):

Table 8 – Roadside Alert Message #3 Contents

Item	Human Value	Machine Value / Units
Lat	34.0833330	0x14-50-B4-32 1/10th micro degrees[56]
Long	-117.8666660	0xB9-BE-F9-5C 1/10th micro degrees
Key Phrase	"movie filming"	0x0F10 (3856)
Other Phrases	"in the downtown area"	0x1F12 (7954)
	"closed to traffic"	0x0301 (769)
	"police directing traffic"	0x1B0F (6927)
	"we are grateful for your cooperation"	0x1A15 (6677)
Heading Applied	Every direction	All bits set, 0xFFFF
Extent	For 500 meters	0x05
Further Info Link	01234	0x04D2

The resulting payload encoding for this content is then the 59 bytes shown below. The details of the T-L-V encodings are as explained before.

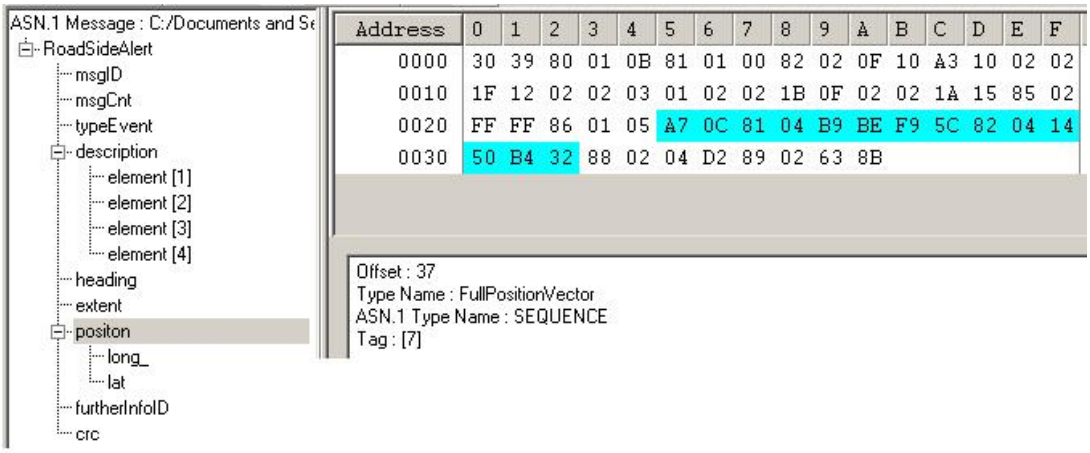

Figure 22 – Roadside Alert Message #3 Decoded

The uses of the `RoadSideAlert` message are not limited to describing events that occur on the roadway; rather all sorts of wide area events can be expressed. This is particularly true for weather related events. A primary use of the ITIS codes is to relate weather conditions to travelers. Consider the following wide area weather event, which would likely be broadcast over a relatively wide area from multiple RSUs.

[56] We will use these location values for the remaining examples to aid the reader in decoding the bytes.

Table 9 – Roadside Alert Message #4 Contents

Item	Human Value	Machine Value / Units
Lat	34.0833330	0x14-50-B4-32 1/10th micro degrees[57]
Long	-117.8666660	0xB9-BE-F9-5C 1/10th micro degrees
Key Phrase	"tornado"	0x1401 (5121)
Other Phrases	"alert cancelled"	0x1B7C (7036)[58]
Heading Applied	Every direction	All bits set, 0xFFFF
Extent	For 500 meters	0x05

The resulting encoding, consisting of 43 bytes is:

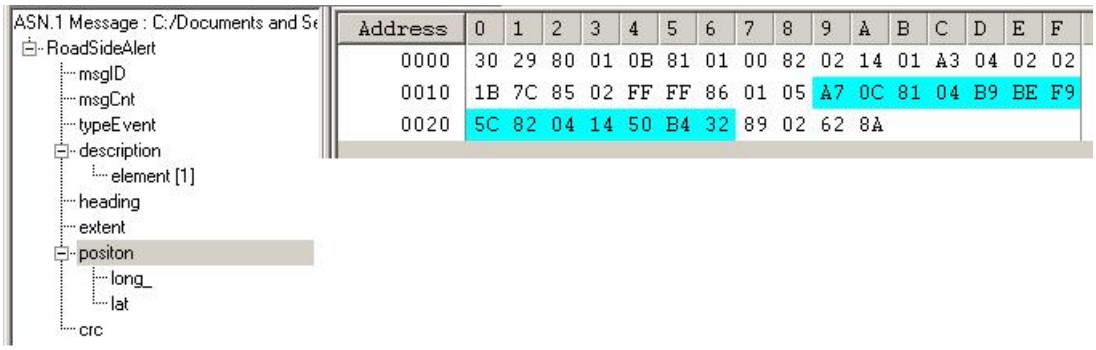

Figure 23 – Roadside Alert Message #4 Decoded

The above example also illustrates that it is important to decode all of the ITIS codes in the message, not just the first one. Often codes are combined in an *event-code* then *status-code* form to create more complex meanings.

Another example of weather use with the ITIS codes is used in localized warnings, such as an icy bridge condition. This message is typically composed of *ice build-up* (code 5910, hex 0x1716) and *on bridge* (code 7937, hex 0x1F01). Such a message is likely to be transmitted from the local roadside infrastructure over a short range and does not need many of the optional elements in the message. The payload bytes for this would be the following 22 bytes. The two ITIS codes are shown in red.

0x 30 14 80 01 0E 81 01 00 82 02 17 16 A3 04 02 02 1F 01 89 02 D4 4F

As a final weather example, consider the message of flash flood warnings canceled. This message is typically composed of *flash flood* (code 3073, hex 0x0C01) and *warning-canceled* (code 7034, hex 0x1B7A). The payload bytes for this would be the following 22 bytes. The two ITIS codes are again shown in red.

0x 30 14 80 01 0E 81 01 00 82 02 0C 01 A3 04 02 02 1B 7A 89 02 22 4D

The RoadSideAlert message is also used inside of other DSRC messages. The encoding forms are the same, but these other message convey additional specific data element details which are useful in some cases. Consider the need to alert nearby vehicles when a public safety vehicle is responding to a service call. The EmergencyVehicleAlert message addresses this need, and wraps the RoadSideAlert message in some additional content. It is defined as follows:

[57] We will use these location values for the remaining examples to aid the reader in decoding the bytes.

[58] The ITIS codes provide values for *warning*, *watch,* and *alert*, as well as other warning advice terms. Each of these terms also has a corresponding "canceled" or "ended" phrase.

```
-- MSG_EmergencyVehicleAlert (Desc Name)
EmergencyVehicleAlert ::= SEQUENCE {
   msgID           DSRCmsgID,
   id              TemporaryID OPTIONAL,
   rsaMsg          RoadSideAlert,
             -- the DSRCmsgID inside this
                              -- data frame is set as per the
                              -- RoadSideAlert.  The CRC is
                              -- set to a value of zero.
   responseType    ResponseType                  OPTIONAL,
   details         EmergencyDetails              OPTIONAL,
                   -- Combines these 3 items:
                   -- SirenInUse,
                   -- LightbarInUse,
                   -- MultiVehicleReponse,

   mass            VehicleMass                   OPTIONAL,
   basicType       VehicleType                   OPTIONAL,
                              -- gross size and axle cnt

   -- type of vehicle and agency when known
   vehicleType     ITIS.VehicleGroupAffected     OPTIONAL,
   responseEquip   ITIS.IncidentResponseEquipment OPTIONAL,
   responderType   ITIS.ResponderGroupAffected   OPTIONAL,
   crc             MsgCRC,
   ... -- # LOCAL_CONTENT
   }
```

Here the `RoadSideAlert` message is combined with an additional eight elements (some optional) which can be used to relate information about the message issuer. In use, receiving OBUs in other vehicles compare the *position* of the sender with their own motion vector and determine what action to take. Note the use of three ITIS subcategories to further describe the sender's vehicle classification.

The *EmergencyDetails* data element is a compound field. It is in fact made up of three 2 bit fields as defined below This is typical of the byte packing used with DSRC to take advantage of the DER encoding rules.

```
EmergencyDetails ::= INTEGER (0..63) -- in one byte
   -- First two bit (MSBs) set to zero.
   -- Combining these 3 items in the remaining 6 bits
   -- sirenUse        SirenInUse
   -- lightsUse       LightbarInUse
   -- multi           MultiVehicleReponse
```

There are many possible ways to describe the case of an emergency vehicle with its lights and sirens active using the ITIS codes; however, some conventions have been developed over the years. The general category of event is "UnusualDriving" and included in that group is: "emergency vehicles on roadway"; this is the key event.[59] Other codes can be used to provide more details.

One can use the `ResponderGroupAffected` group subcategory to tell which general type of vehicle is involved (*police units*, *fire units*, *ambulance units*, etc., or the generic *emergency vehicle units*). This can be placed in the ITIS code sequence or in the `responderType` element. One can use the `IncidentResponseEquipment` group subcategory to tell which specific type of vehicle is involved (this list comes from the fire service and from transportation sources, and has items like BLS for Basic Life Support, a minimally equipped ambulance as well as simple *ambulance*) when that level of detail is needed. In this example, both are used simply to illustrate.

[59] Other codes in that category may also apply; note that these include: *high speed emergency vehicles*, and *emergency vehicle warning cleared*. In the Advice-Instructions-Mandatory group are phrases like: *allow emergency vehicles to pass* which may be of use as well.

Vehicle mass and type (an axel count value) can also optionally be sent, in this example a mass of 2500 kg was used.

And a well-formed example of using the `EmergencyVehicleAlert` message could have the following data elements in it:

Table 10 – Emergency Vehicle Alert Message #5 Contents

Item	Human Value	Machine Value / Units
Lat	34.0833330	`0x10-A0-0E-68` 1/10th micro degrees[60]
Long	-117.8666660	`0xB9-BE-F9-5C` 1/10th micro degrees
Speed	123 m/s in 0.02 units plus trans	`0x300C` (6150) in 0.01 m/s
Heading in FPV	due north	`0x00` (0)
Key Phrase	"emergency vehicles on roadway"	`0x5806` (`Bx010` + 1795)
Heading Applied	East and westbound traffic	Four bits set, `0x1818`
Extent	For 50 meters	`0x03`
Mass	2500 kg	`0x32` (50)
Response Equip	ambulance	`0x0F10` (10086)
Responder Type	ambulance units	`0x0F10` (9734)
Response Type	'01' = emergency	`0x01`
Response Details	Complex byte representing	`0x29`
SirenInUse	'10' = inUse	
LightbarInUse	'10' = inUse	
MultiVehicleResponse	'01' = singleVehicle	

The *responseType* element becomes the fourth tag in the outer sequence (`0x83`), while the *details* is the sixth tag and the *mass* is the seventh, and the *responderType* element follows as the tenth tag the normal way.

Note that the *msgID* for this message is `0x05`, while the *rsaMsg* (the name of the Roadside Alert in this case) continues to have its own *msgID* of `0x0E` and its own *CRC* values (set to zero here). Within the *alertMsg*, the decoding process for the encapsulated message is exactly the same as previously shown for any other `RoadSideAlert` message.

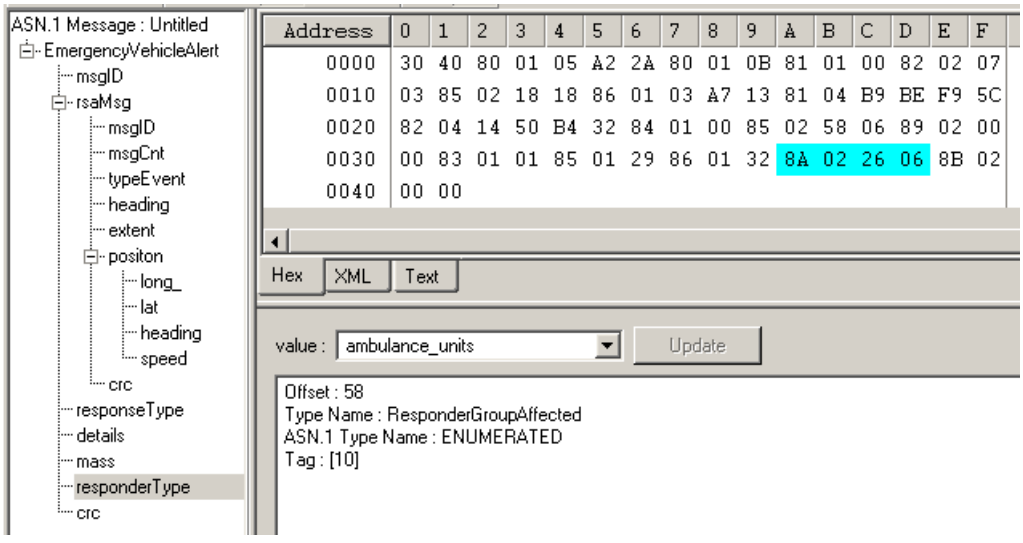

Figure 24 – Emergency Vehicle Alert Message #5 Decoded

[60] We use these location values for the remaining examples to aid the reader in decoding the bytes.

The previous examples show how the ITIS codes and the other objects defined in the `RoadSideAlert` message are used and encoded. The `RoadSideAlert` message is a widely used workhorse in the DSRC messaging system for describing most any event needed.

4.7 Relevance to ATIS or ITIS Message

As the previous examples should indicate, this particular DSRC message shares a great deal of content with the event message of ATIS. The use of the ITIS codes allows similar descriptive expressions and encoding of the same content. Translation between the two is therefore simplified as well. A simple linking method is provided in the DSRC message to connect the terse DSRC message back to the more verbose ATIS form. The key to all of this is the use of ITIS, a set of uniform codes developed with over 20 years of operational experience in the US and Europe.

It is sometimes very helpful to list the categories of the ITIS phrase categories that can become the key event (not every category can). This is provided below (text adapted from the ATIS event guide). Note the large emphasis on weather related events, a major influence on driving conditions. Many non-roadway events (such as a local ball game) which can have an impact on traffic conditions also need to be described and are listed, as well as general transit information. Note the section on vehicle or traveler class. This allows creating and describing events and restrictions that pertain only to a class of user (i.e., trucks with trailers, carpool lane users, etc.). And while all of the items in the below groups can be used to provide the "key" event, they are also part of the normal ITIS codes so that they can be easily mixed with other ITIS phrases to fully describe any event as needed.

Traffic and Roadway Related
- Accidents and Incidents
- Alternate Routes
- Closures and Roadwork
- Delay Status Cancellation
- Device Status
- Mobile Situations
- Obstructions
- Traffic Conditions
- Unusual Driving

Non-Roadway Events Related
- Disasters
- Disturbances
- Special Events
- Sporting Events

Transit and Parking Related
- Parking Information
- Transit Modes
- Transit Operations

Weather Related
- Pavement Conditions
- Precipitation
- Temperatures
- Visibility and Air Quality
- Weather Conditions
- Winds
- Winter Driving Index
- Winter Driving Restrictions

Vehicle or Traveler Class Related
- Advice/Instructions Mandatory
- Advice/Instructions Recommendations
- Incident Response Equipment
- Responder Group Affected
- Restriction Class
- Suggestion Advice
- Traveler Group Affected
- Vehicle Group Affected
- Warning Advice

Those categories which cannot be used as a key event are those typical of a supporting or a descriptive nature and would not make sense as a standalone code. These categories are:

Categories which can not be used as the key phrase item:
- Asset Status
- Generic Locations
- Incident Response Status
- Lane Roadway
- Objects
- Qualifiers
- Roadside Assets
- System Information
- Units

In addition to the above "classic" categories, a number of new categories have been added to the ITIS codes during the time this guide was in development These deal with the new groups needed to fully reflect MUTCD road sign types in the codes, and they are shown below. Of the below ten categories, only the *RegulatoryAndWarningSigns* group can be used as a key phrase category (as a basic event type); the others must be used with other selections.

Added in Support of MUTCD:

- Large Numbers
- MUTCD Locations
- Named Objects
- Recreational Objects And Activities
- Regulatory And Warning Signs
- Small Numbers
- States And Territories
- Street Suffixes
- Structures
- Valid Maneuvers

When translating ITIS codes to their textual equivalents, it is vitally important to use the precise strings defined in the standard if these representations will be sent to other users of ITS standards for additional processing. If the end user is a textual display or other form of human readable device, this is less important.[61] It is typical to run the phrases into a function to convert them into more proper English sentences. Such routines (beyond the scope here) typically capitalize the first letter and may add additional bits of grammar and punctuation to the resulting string.

Deployments can find the exact members and spelling of each of these groups in the ASN or XML listings for the ITIS standard. This is all part of the ITIS standard (which is maintained by the SAE ATIS committee with close coordination with the other SDOs). You may also find the ITS standards forum area devoted to *ITIS codes* helpful.

4.8 Interface Design Issues/Considerations

The interface design issues with this message revolve more around human factors and display issues than the other applications in the guide. Solving these are clearly out of scope for a message set guide, but we may safely touch of a few of the issues.

How the resulting message will be used or displayed varies with the subjective importance of the message content. It may affect how the recovered message is routed (to different devices). For example, a speed limit sign message may not be displayed at all unless the vehicle is exceeding the posted limit and other priorities or demands for display are low. Some messages are immediate, other vary with speed, still others apply for differing lengths of time (or distances of travel). The type of alerting may vary by content and by platform level (luxury vehicles are expected to have a richer user experience than low-end ones). All of these are complex issues that a real world design must come to grips with.

One item that can be discussed because it probably must exist[62] in any design is how the ITIS codes are translated to other forms for use. In the code fragment example given here, we simply translate a 2 byte code value into its string equivalent form. Any real application would take this a bit further, likely creating a more polished presentation of the string used into proper English. In a similar fashion, any message encoding device has to deal with the problem of how to reasonably present to the end user a selection of over 2000 possible codes to pick from. Most approaches try to break this down into basic categories and hide from the user most of the possible choices until some basic data about the event is known. We have more or less skipped this step in the application examples.

[61] Some automotive developers have devised translation maps to represent the various codes and categories into a basic set of icons. Others have devised short text to sound or voice translation systems that are used to create audible warnings. In the dynamic environment of a vehicle cockpit, long text messages are not ideal, and only high end vehicles have the display means to show such text.

[62] The alterative would be multiple copies of the code doing more or less the same thing each time; each having to be as large as all the possible text strings found in ITIS, which is a modestly large collection of text to duplicate, about 150k in overall size.

While it is quite possible to build a reusable class of objects to deal with the ITIS codes and whatever other media they will be expressed in, this too is out of scope. For simple instruction purposes, the code fragment provides a class of functions to translate ITIS codes into strings. Because the number of codes used in this message are fairly short,[63] we can pack all the codes into a short table structure and then hand this off to the class to fill out for us.

Consider a simple structure made up of :

```
typedef struct ITISentry  {
   SHORT    code;      // the 2 byte itis code
                       // (upper byte is category , lower byte is item)
   CString  text;      // the enumeration string for this code
                       // one could use a pointer here as well
   BOOLEAN  alert;     // an indication to alarm when this code is used
   }
```

If we have a pointer to an array of these, we can pack our ITIS codes into the first element, then call the ITIS handler class to fill in the rest of the structure (the string and alarm values) for us. We can then build a final function to translate this into a rough but readable English sentence, an XML expression form, or any other format we need. The source code fragments illustrate code to do this; consult the source listing for more details. In a production environment, such a handler class could be greatly expanded, and would form a support module that could be used by multiple applications to deal with ITIS codes. This concept is illustrated in the image below.

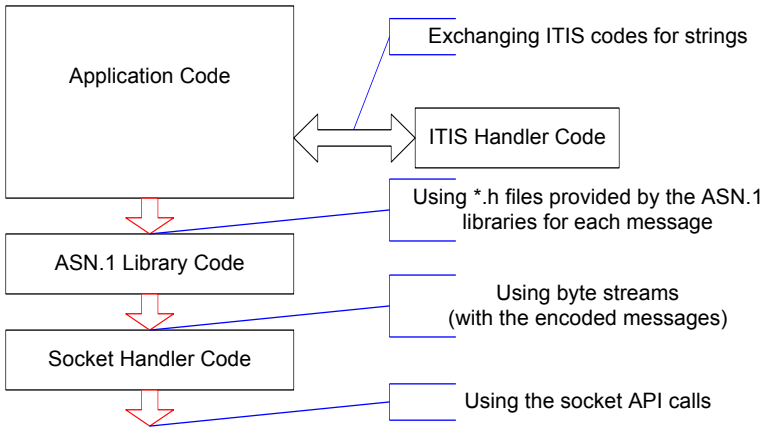

Figure 25 – Protocol Stack Used in Examples with ITIS

4.9 Message Sharing Issues/Considerations

This message is unlike the BSM where multiple tasks may want to look at a single instance of a message This single message is used by multiple different applications, some of which embed the message inside another one. Here we have a more classic reuse of the message decoding functionality. It is reasonable to presume that any application with a need to decode the `EmergencyVehicleAlert` message (which has inside it a copy of the `RoadSideAlert` message) might call the application that deals with the `RoadSideAlert` message for decoding services.

As mentioned in the design issues subsection above, the decoder developed for translating the ITIS codes into strings or other expression is a natural collection of functionality quite suitable for sharing between applications and messages.

[63] The ATIS event message allow as many as 100 codes to be used in a single message, allowing the (remote) possibility of a huge structure. Here we are limited to no more than 9 possible codes and typically have less.

5. Application 3
Probe Vehicle Reporting

5.1 Description

This application deals with the delivery of historical vehicle data sets (referred to as probe data) collected by each vehicle OBU which is delivered to an RSU when passing within its range. The structure and form of the passed data is the primary topic in this application. The DSRC message set also provides for a set of control messages from the RSU to be sent to the OBU to control what data is collected and reported. These control messages (`ProbeDataManagement` messages) are not covered in this edition of the guide, so the data collection process reflects in the default behavior in the absence of any management messages. These probe data elements include:

Acceleration
Ambient Air Pressure
Ambient Air Temperature
Antilock Brake Status
Brake Applied Pressure
Brake Boost Applied
Coefficient Of Friction
Date
Driving Wheel Angle
Elevation
Exterior Lights
Heading
Latitude
Longitude
Obstacle Direction
Obstacle Distance
Probe Segment Number
Rain Sensor
Speed
Stability Control Status
Steering Wheel Angle
Steering Wheel Angle Rate Of Change
Sun Sensor
Time
Tire Pressure
Tire Pressure Threshold Detection
Traction Control State
Vehicle Type
Vertical Acceleration
Wiper Rate
Wiper Status Front
Wiper Status Rear
Yaw Rate

Also included may be some confidence data and other supporting data elements.

Over 100 use cases have been identified and it was from these use cases that the data elements have been derived. Not all vehicles have all these data; each vehicle provides only the information for which it has data. The collection of these data at one point in time is referred to as a snapshot. Snapshots are generated in three manners:

- **Periodically** – at intervals based on vehicle movement between RSUs
- **Event Triggered** – these occur when the state of certain vehicle status elements change
- **Starts and Stops** – these occur when a vehicle starts moving and stops moving

The intent behind the snapshots is to obtain ubiquitous coverage. However there are limitations in the data storage in the vehicles (30 snapshots) and RSUs are widely spaced in rural areas and closer together in urban areas. To compensate for this, the distances between snapshot intervals varies by speed, being larger at high speed and smaller at low speed.

This sampling variation is illustrated in the graphic below that was taken from test data in the Detroit area. It shows the wider spacing on the higher speed roads. An algorithm inside the OBU handles the process of selecting which points are to be sent to the RSU.

The `ProbeVehicleData` (PVD) message is used to relate this stored information back to the local RSU over a service channel. In general, all OBUs are collecting probe data when the vehicle key is on.

The OBU becomes aware of the presence of an RSU that accepts probe data from hearing a Wave Service Announcement (WSA) that describes the probe application. The OBU uses data sent in the Provider Service Announcement (PSA) to determine which service channel to switch to and uploads its current data snapshots to that RSU.

The probe data includes an anonymous identifier, the Probe Segment Number (PSN). This is used by the application to determine the trajectory of vehicles for applications such as turning movement counts. The PSN is regularly changed to ensure privacy. This change occurs following either 120 seconds or 1 km, whichever comes last. To aid anonymous snapshots and prevent association between consecutive PSNs, a variety of mechanisms are used including not transmitting the same PSN to more than one roadside unit, randomizing time and distance when changing to a new PSN, limiting the duration to 120 seconds or 1 km (whichever comes last), and other logical mechanisms to ensure privacy.

Figure 26 – Probe Vehicle Data Location Reports

The standard recommends a specific mechanism for removing snapshots from memory when the message buffer is full and snapshots are still being generated. This involves queuing the snapshots in the order received and repeatedly deleting alternates starting with the oldest, with the exception of keeping the very oldest. This oldest one provides information for travel times. Deleting alternates keeps the data retrieval as geographically broad as possible.

5.2 Relevance to IntelliDrive℠

The National System Requirements (NSR) describes the VII system and the services it provides. The current initiative within USDOT is labeled IntelliDrive℠. This encompasses a broader range of products that are designed to encourage multiple commercial enterprises to collaborate in achieving the broad goals of connecting vehicles and the infrastructure. The NSR is substantial and deals with all aspects of an installed system; this section is concerned only with the probe data collection process.

The roadside infrastructure user is part of the system that interacts with the roadside units to gather a range of information that can include probe data. The users of probe data include a wide range of use cases beyond the ATMS. Under the current approach by USDOT, the Probe Data Service (PDS) that is the component of the NSR will likely be a commercial operation. For the DSRC developer, the data connection beyond the roadside equipment is thus currently undefined.

5.3 Communication Parameters

Probe Vehicle Reporting

Minimum Required Communication Range	Range of passing RSU
Message Direction	Bi-directional exchange along with a WSA and PSC exchange
Broadcast Interval	NA
Priority Assignment Criteria	The PDM and PVD messages are both assigned the same PSID value; the PDM message is accounted with a WSA as well

5.4 How to Use the Message

This application section is primarily concerned with message considerations and the processing of position data. An active OBU is constantly collecting and discarding various data sets (snapshots) for use in the probe message which involves a complex *gather-discard* process that is not considered here (see the J2735 annex for further details of these algorithms). The selection of *which* snapshot is to be sent is a function of this; here we concern ourselves with the mechanical details of encoding the chosen data only.

The probe data message itself (called `ProbeVehicleData`) is shown below. It consists of the normal message type identifier, `DSRCmsgId`, followed by either a `ProbeSegmentNumber` (used to identify the RSUs coverage area[64]), or a `VehicleIdent` (used to identify public fleet vehicles).

```
ProbeVehicleData ::= SEQUENCE {
   msgID           DSRCmsgID,             -- App ID value, 1 byte
   segNum          ProbeSegmentNumber OPTIONAL,
                                          -- a short term ident value
                                          -- not used when ident is used
   probeID         VehicleIdent OPTIONAL,
                                          -- ident data for selected
                                          -- types of vehicles
   startVector     FullPositionVector,    -- the space and time of
                                          -- transmission to the RSU
   vehicleType     VehicleType,           -- type of vehicle, 1 byte
   cntSnapshots    Count OPTIONAL,
                                          -- a count of how many snapshots
                                          -- type entries will follow
   snapshots       SEQUENCE (SIZE(1..32)) OF Snapshot,
                                          -- a seq of name-value pairs
                                          -- along with the space and time
                                          -- of the first measurement set
   ... -- # LOCAL_CONTENT
   } -- Est size about 64 bytes plus snapshot sizes (about 12 per)
```

The next data concept to be sent in the message is the `startVector` of the vehicle (the position, speed, heading, and time of the vehicle when it initiated a message transmission to the in-view RSU) . This data value is limited to the reception range of the RSU, but speed and heading are used to determine the direction of travel of the responding OBU.

This is followed by a `VehicleType` data element used to define the classification of the reporting vehicle in terms of its FHWA category (which is required for the states to report vehicle flow).

[64] The PSN is received by the OBU when it processes the PSA for that RSU and learns what service channel is to be used to send to the RSU. The PSN is then echoed back to the RSU in the body of the `ProbeVehicleData` message.

Next, a `cntSnapshots` contains the number of snapshots to follow. This can also be determined by parsing the actual snapshots in the ASN package, but it is provided here as an optional courtesy element.

After this comes the snapshot data itself, the critical payload of the message. From one to 32 snapshot data items can be enclosed in the message. The number present is also dependent on the resulting message length, which is limited in most lower layers. We now consider the possible contents of each snapshot.

```
Snapshot ::= SEQUENCE {
   thePosition  FullPositionVector,
                -- data of the position and speed,
   safetyExt    VehicleSafetyExtension OPTIONAL,
   datSet       VehicleStatus          OPTIONAL,
                -- a seq of data frames
                -- which encodes the data
  ... -- # LOCAL_CONTENT
   }
```

Snapshots consist of a position data frame (always) and then either a `VehicleSafetyExtension` data frame and/or a `VehicleStatus` data frame with further additional information acquired at that position point. Typically only one single item is selected from the support list to be sent in the snapshot. The position represents when that associated data (if any) was taken or observed.

In the absence of directed data from a probe management message, the snapshot consists simply of the position data from a past location of the vehicle (also called a *breadcrumb* when used in other messages). Each of the two data frames is discussed in turn below.

```
VehicleSafetyExtension ::= SEQUENCE {
   events             EventFlags     OPTIONAL,
   pathHistory        PathHistory    OPTIONAL,
   pathPrediction     PathPrediction OPTIONAL,
   theRTCM            RTCMPackage    OPTIONAL,
   ... -- # LOCAL_CONTENT
   }
```

The `VehicleSafetyExtension` data frame contains some of the various items that the snapshot can report back. Along with the `VehicleStatus` data frame, the items defined here represents the full range of data concepts that can be reported in the snapshot. The `VehicleSafetyExtension` data frame is less often used in snapshots[65] than the contents of the `VehicleStatus` data frame, but it contains:

Events	Currently active event flag information
Vehicle Path History	Recent vehicle path points (breadcrumbs)
Vehicle Path Predictions	A predicted vehicle path
Various RTCM Messages	GPS correction information and phase data

The `VehicleStatus` data frame is discussed next.

```
VehicleStatus ::= SEQUENCE {
   lights          ExteriorLights OPTIONAL,                -- Exterior Lights
   lightBar        LightbarInUse  OPTIONAL,                -- PS Lights

   wipers   SEQUENCE {
         statusFront    WiperStatusFront,
         rateFront      WiperRate,
         statusRear     WiperStatusRear       OPTIONAL,
         rateRear       WiperRate             OPTIONAL
         } OPTIONAL,                                       -- Wipers
```

[65] This information is more often found in the Part II section of the Basic Safety Message (BSM)

```
brakeStatus  BrakeSystemStatus OPTIONAL,
                -- 2 bytes with the following in it:
                --   wheelBrakes        BrakeAppliedStatus,
                --                      -x- 4 bits
                --   traction           TractionControlState,
                --                      -x- 2 bits
                --   abs                AntiLockBrakeStatus,
                --                      -x- 2 bits
                --   scs                StabilityControlStatus,
                --                      -x- 2 bits
                --   brakeBoost         BrakeBoostApplied,
                --                      -x- 2 bits
                --   spareBits
                --                      -x- 4 bits
                -- Note that is present in BSM Part I
                                                        -- Braking Data
brakePressure   BrakeAppliedPressure  OPTIONAL,         -- Braking Pressure
roadFriction    CoefficientOfFriction OPTIONAL,         -- Roadway Friction


sunData         SunSensor             OPTIONAL,         -- Sun Sensor
rainData        RainSensor            OPTIONAL,         -- Rain Sensor
airTemp         AmbientAirTemperature OPTIONAL,         -- Air Temperature
airPres         AmbientAirPressure    OPTIONAL,         -- Air Pressure

steering   SEQUENCE {
      angle      SteeringWheelAngle,
      confidence SteeringWheelAngleConfidence   OPTIONAL,
      rate       SteeringWheelAngleRateOfChange OPTIONAL,
      wheels     DrivingWheelAngle              OPTIONAL
      } OPTIONAL,                                       -- steering data

accelSets  SEQUENCE {
      accel4way       AccelerationSet4Way           OPTIONAL,
      vertAccelThres  VerticalAccelerationThreshold OPTIONAL,
                                                    -- Wheel Exceeded point
      yawRateCon      YawRateConfidence             OPTIONAL,
                                                     -- Yaw Rate Confidence
      hozAccelCon     AccelerationConfidence        OPTIONAL,
                                                     -- Acceleration Confidence
      confidenceSet   ConfidenceSet                 OPTIONAL
                                                    -- general ConfidenceSet
      } OPTIONAL,


object     SEQUENCE {
      obDist          ObstacleDistance,          -- Obstacle Distance
      obDirect        ObstacleDirection,         -- Obstacle Direction
      dateTime        DDateTime                  -- time detected
      } OPTIONAL,                                -- detected Obstacle data



fullPos         FullPositionVector OPTIONAL,     -- complete set of time and
                                                 -- position, speed, heading
throttlePos     ThrottlePosition OPTIONAL,
speedHeadC      SpeedandHeadingandThrottleConfidence OPTIONAL,
speedC          SpeedConfidence OPTIONAL,

vehicleData   SEQUENCE {
      height        VehicleHeight,
      bumpers       BumperHeights,
      mass          VehicleMass,
      trailerWeight TrailerWeight,
```

```
      type          VehicleType
      -- values for width and length are sent in BSM Part I as well.
      } OPTIONAL,                                  -- vehicle data

vehicleIdent   VehicleIdent OPTIONAL,              -- comm vehicle data

j1939data      J1939data OPTIONAL,           -- Various SAE J1938 data items

weatherReport SEQUENCE {
      isRaining        NTCIP.EssPrecipYesNo,
      rainRate         NTCIP.EssPrecipRate       OPTIONAL,
      precipSituation  NTCIP.EssPrecipSituation  OPTIONAL,
      solarRadiation   NTCIP.EssSolarRadiation   OPTIONAL,
      friction         NTCIP.EssMobileFriction   OPTIONAL
      } OPTIONAL,                                  -- local weather data

gpsStatus      GPSstatus          OPTIONAL,      -- vehicle's GPS

... -- # LOCAL_CONTENT OPTIONAL,
}
```

The `VehicleStatus` data frame contains various items that the snapshot can also report back. In the absence of `ProbeDataManagement` messages to control the reporting for specific items, this area of the message is not used. When one or more of these items is requested in a `ProbeDataManagement` message (along with various limits and reporting rates) then that item may be returned in a subsequent snapshot. The data content found here is wide ranging and contains information about:

Air Temp and Pressure	The average measured air temperature and pressure
Brake Status	The state of brakes and associated traction, control, and safety systems
Detected Roadway Objects	The position of objects about which the vehicle was forced to swerve to avoid
GPS Status	Gross status of GPS system
Identification Labels	VIN, Fleet ID, and Owner Data for public fleets
Lights	Vehicle lighting and any public safety lighting being used
Physical Characteristics	Vehicle size, bumper placement, mass, trailer data, and gross classification data
Rain Rates	The average measured rainfall rate
Road Friction	The average measured surface road friction
Sun Radiation	The average measured sun radiation rate
Vehicle Accelerations	Acceleration in 3 dimensions and yaw rate
Vehicle Speeds	Speed, throttle, and heading data
Vehicle Steering	Steering angles and rates of change
Weather Information	Rainfall rates, solar radiation data, road friction
Wipers	The state of wiper systems and the run rates

Note that a number of these relate to localized weather information and are expected to be used to gather local weather conditions. Not all vehicles are expected to be able to report on all the items defined here. Vehicles which lack the requested data (or do not implement that probe ability) simply do not response to `ProbeDataManagement` messages of that type.

5.5 Representative Encoding of the Message

Considering the output from the ASN1c tool which represents this message (which you can find in the code sections or by running the source listing of the standard into the tool yourself), we have:

```
/* ProbeVehicleData */
typedef struct ProbeVehicleData {
    DSRCmsgID_t          msgID;
    ProbeSegmentNumber_t    *segNum             /* OPTIONAL */;
    struct VehicleIdent *probeID                /* OPTIONAL */;
    FullPositionVector_t    startVector;
    VehicleType_t       vehicleType;
    long                     *cntSnapshots      /* OPTIONAL */;
    struct snapshots {
        A_SEQUENCE_OF(struct Snapshot) list;

        /* Context for parsing across buffer boundaries */
        asn_struct_ctx_t _asn_ctx;
    } snapshots;
    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _asn_ctx;
} ProbeVehicleData_t;
```

The general format of the above should now be familiar from the prior two applications. The two entries listed below are treated precisely like their former instance were, and will not be considered further (see the other two applications for detailed explanations of how to encode and use these items). Note that both are required instances, so a pointer is not used.[66]

```
DSRCmsgID_t      msgID;
                    // the type of message, a byte, treated like
                    // the prior two applications were
FullPositionVector_t    startVector;
                    // a 3D position, speed, heading, time value,
                    // treated like it was in the RSA message
```

The below three items are simply integers and are treated as outlined in prior sections. Note that two of these use pointers and therefore the application must allocate storage for them before storing a value.

```
ProbeSegmentNumber_t    *segNum
                    // a value which we can treat as a
                    // simple constant
VehicleType_t       vehicleType;
                    // a value which will be constant for any
                    // given OBU
long                *cntSnapshots
                    // An integer count of the snapshots to follow
```

In cases where we have a public fleet type of vehicle, the `VehicleType_t` is not sent, rather the `VehicleIdent` structure is sent. This is a complex nested data structure defined (by way of a pointer) as:

```
/* VehicleIdent */
typedef struct VehicleIdent {
    DescriptiveName_t    *name          /* OPTIONAL */;
    VINstring_t      *vin               /* OPTIONAL */;
    IA5String_t      *ownerCode         /* OPTIONAL */;
    TemporaryID_t *id                   /* OPTIONAL */;
    VehicleType_t *vehicleType          /* OPTIONAL */;
    struct vehicleClass {
```

[66] Recall from the prior discussion that pointers are used in this tool as a means to indicate an optional element. A null pointer indicates the absence of the optional element. And as discussed before, you are responsible for allocating storage space for whatever the pointer points at when there is an instance of it.

```
        vehicleClass_PR present;
        union VehicleIdent__vehicleClass_u {
            VehicleGroupAffected_t     vGroup;
            ResponderGroupAffected_t      rGroup;
            IncidentResponseEquipment_t   rEquip;
        } choice;

        /* Context for parsing across buffer boundaries */
        asn_struct_ctx_t _asn_ctx;
    } *vehicleClass;

    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _asn_ctx;
} VehicleIdent_t;
```

The four elements [`DescriptiveName, VINstring, IA5String, and TemporaryID`] (all referenced using pointers) are each simply a string and can be handled as outlined before.

The `VehicleType` is an enumeration list and also handled as denoted before.

The `vehicleClass` is a choice of three different enumeration lists and is handled as denoted before.

The only really novel and new item in any of this is the set of `snapshots` itself. Note that the code indicates a "list" of items of type "`Snapshot`" which the ASN1c tool has created for us as its way to manage a list.

```
struct snapshots {
    A_SEQUENCE_OF(struct Snapshot) list;
```

This is a bit tricky to use, one of reasons being that the ASN1c tool does not seems to document very well how it is expected to work. If you examine the source code in the supporting file `ASN_SEQUENCE_OF.h`, you will quickly discover that the tool has provided a set of helper functions to add and remove items to/from this list.

The function `#define ASN_SEQUENCE_ADD(headptr, ptr)` is to be noted. This routine takes a passed pointer (to the type required) and appends it to the list object (the item marked *headptr* here).[67] An *empty* routine is also provided. In practice, this appears in our application code as a fragment like:

```
if (there are snapshots to add)
{
    // allocate memory and fill it with member var and add
    for (i=1; i<cntSnapshots; i++)
        {
        // create memory for snapshot and its parts, snap[]
        // assign content values to it as needed
        asn_sequence_add(theList, snap[i]);  // append to list
        }
    theMsg->description =
        (struct ProbeVehicleData::snapshots *) theList;
        // attach list to main msg
}
```

Once a snapshot has been created (and added to the list), then its component parts can be filled out (if not already done as part of the creation process). As noted before, `thePosition` is always required, and then (optionally) either the `safetyExt` (`VehicleSafetyExtension`) or `datSet` (`VehicleStatus`) elements with whatever content is to be conveyed.

[67] This need to add to lists occurs whenever we have sequence of the same object in the ASN. It occurs several times in the actual standard but is shown implemented in the example code in only two places, here and the ITIS descriptions of the *RoadSideAlert* message. Note that in that case an indirect pointer is used because the data object is `OPTIONAL`, whereas here it is required.

When we are simply relating prior position points back, the process of filling out the data simplifies to the below.

```
Snapshot ::= SEQUENCE {
   thePosition  FullPositionVector,
                -- data of the position, time, and speed,
   }
```

The time value to be sent in the `FullPositionVector` here is worthy of some additional discussion. The standard allows a full time value, and the annex provides no further requirements on which of the optional time elements need to be sent. As a practical note, the year, month, and day data elements need never be sent, as these will be typically be the same as the current time; prior positions do not extend beyond 24 hours. Thus time is reduced to the *hour*, *minute* and *second* at which the breadcrumb data point occurred.

```
DDateTime ::= SEQUENCE {
   year     DYear     OPTIONAL,     -- 2 bytes    Not Used in this case
   month    DMonth    OPTIONAL,     -- 1 byte     Not Used in this case
   day      DDay      OPTIONAL,     -- 1 byte     Not Used in this case
   hour     DHour     OPTIONAL,     -- 1 byte
   minute   DMinute   OPTIONAL,     -- 1 byte
   second   DSecond   OPTIONAL      -- 2 bytes
   }
```

The remainder of the FPV's optional items (shown below) should be completed and sent as a general rule. The various confidence values can be set to zero (`0x00`) when no data is present, or not sent. The `posAccuracy` should relate the then current noise and accuracy data of the GPS system; if this data is not available the default values for no data available (`0xFF,FF,FFFF`) should be used. Of course the lat and long values are always present.

```
FullPositionVector ::= SEQUENCE {
   utcTime             DDateTime OPTIONAL,   -- time with millisecond precision
   long                Longitude,            -- 1/10th micro degree
   lat                 Latitude,             -- 1/10th micro degree
   elevation           Elevation OPTIONAL,   -- 3 bytes, 0.1 m
   heading             Heading OPTIONAL,
   speed               TransmissionAndSpeed OPTIONAL,
   posAccuracy         PositionalAccuracy OPTIONAL,
   timeConfidence      TimeConfidence OPTIONAL,
   posConfidence       PositionConfidenceSet OPTIONAL,
   speedConfidence     SpeedandHeadingandThrottleConfidence OPTIONAL,
   ... -- # LOCAL_CONTENT
   }
```

Or in the C form produced by the ASN1c tool:

```
/* FullPositionVector */
typedef struct FullPositionVector {
       struct DDateTime       *utcTime            /* OPTIONAL */;
       Longitude_t               Long;
       Latitude_t                lat;
       Elevation_t               *elevation       /* OPTIONAL */;
       Heading_t              *heading            /* OPTIONAL */;
       TransmissionAndSpeed_t  *speed             /* OPTIONAL */;
       PositionalAccuracy_t          *posAccuracy  /* OPTIONAL */;
       TimeConfidence_t       *timeConfidence     /* OPTIONAL */;
       PositionConfidenceSet_t*posConfidence      /* OPTIONAL */;
       SpeedandHeadingandThrottleConfidence_t  *speedConfidence
                                                  /* OPTIONAL */;
       /* Context for parsing across buffer boundaries */
       asn_struct_ctx_t           _asn_ctx;
} FullPositionVector_t;
```

The supporting application provides another way to build up PVD messages. Like the other two applications already covered, there is a dialog based data entry system that can build and decode the ASN. The information found Table 11 can be used (or any other valid data) to build up such message for test and debug use.

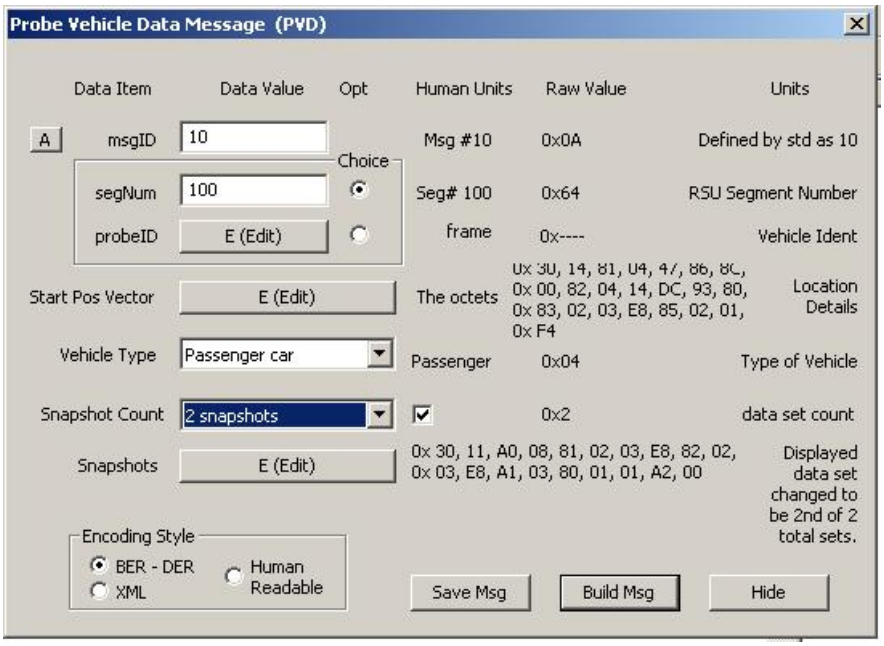

Figure 27 – PVD Application, Main Window View

Each button expands into further dialog; here the position dialog is shown.

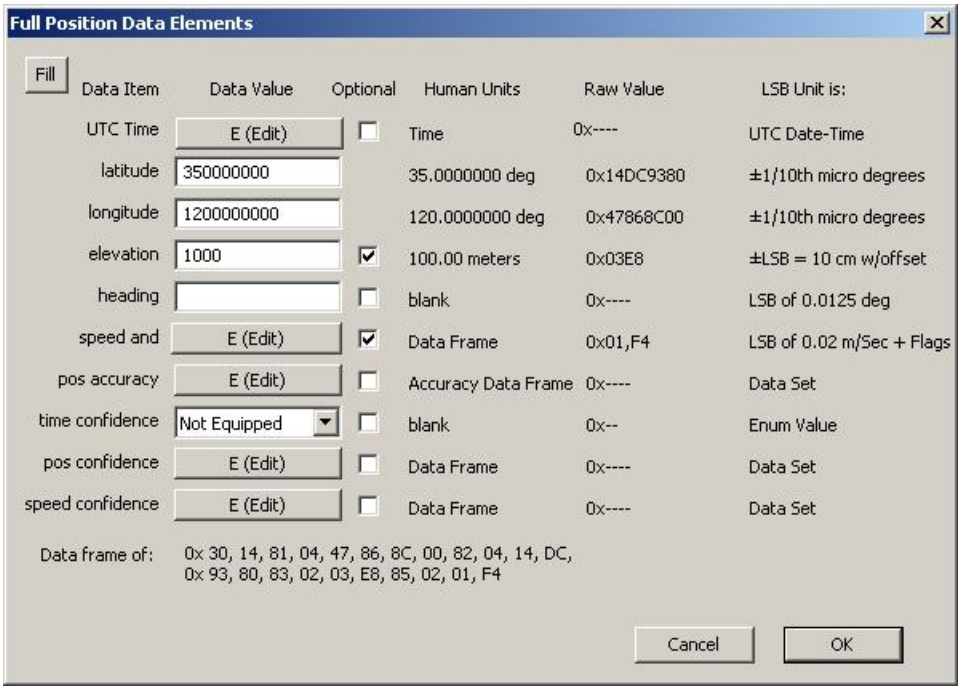

Figure 28 – PVD Use of the Full Position Dialog

Snapshot data (position as well as various data values) can also be added to the message.

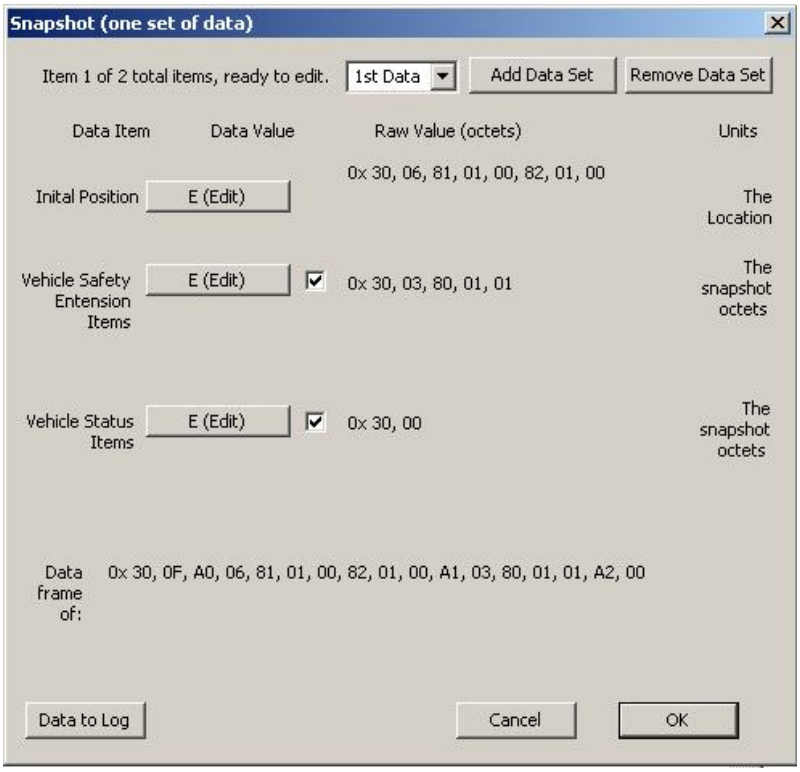

Figure 29 – PVD Snapshots Data View

5.6 Examples of Well-Formed Messages

The simplest `ProbeVehicleData` message possible is shown below (again using the ASN1c tool). A segment number of zero and two snapshot values are present. Both sets of lat-long values are set to zero; other optional content is not sent. This is short and completely legal, but not particularly representative of a real message. The reader should be able to parse over the T-L-V elements at this point (if not, refer to prior sections).

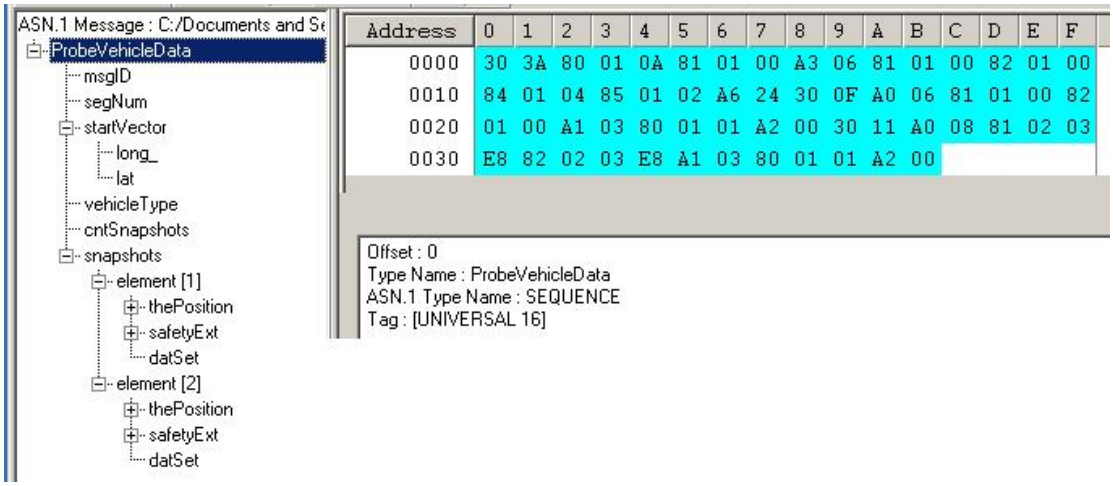

Figure 30 – Trivial PVD Message Encoding

We now need to strip in some real world data (values shown below, which you may recall seeing in the prior messages encodings):

Table 11 – Data Elements in a Valid PVD Message

Element Name	Human Value	Byte Length	Hex Value	Comments
msgID	10	1	0x0A	Defined by std
Segment Number	100	1	0x64	Arbitrary
Start Vector				Built up of:
Long	-120.0000 deg west	4	0xB8797400	1/10th micro degree
Lat	+35.0000 deg north	4	0x14DC9380	1/10th micro degree
Elevation	1000 meters	2	0x2710	
Vehicle Type	Car		0x4	
Count of Snaps	1		0x01	
Snapshot # 1				Built up of:
UTC Time				Built up of:
hour	12	4	0x0C	
minute	05	4	0x05	
second	06.000	2	0x1770	
long	-120.0001 deg west	4	0xB8797018	1/10th micro degree
lat	+35.0001 deg north	4	0x14DC9768	1/10th micro degree
elevation	1000 meters	2	0x2710	
heading	Due South	2	0x3840	
transmission	forward	-	0x4 (upper bits)	Defined by std
speed	50.00 m/s	2	0x09C4 -> 0x49C4	in 0.02 m/s

And then the DER encoded message becomes as shown below. All the rules which the prior applications have explained for DER encodings and the T-L-V patterns also apply here (see prior applications, section 3. and 4., and also section 6. for detailed explanations of decoding these bytes).

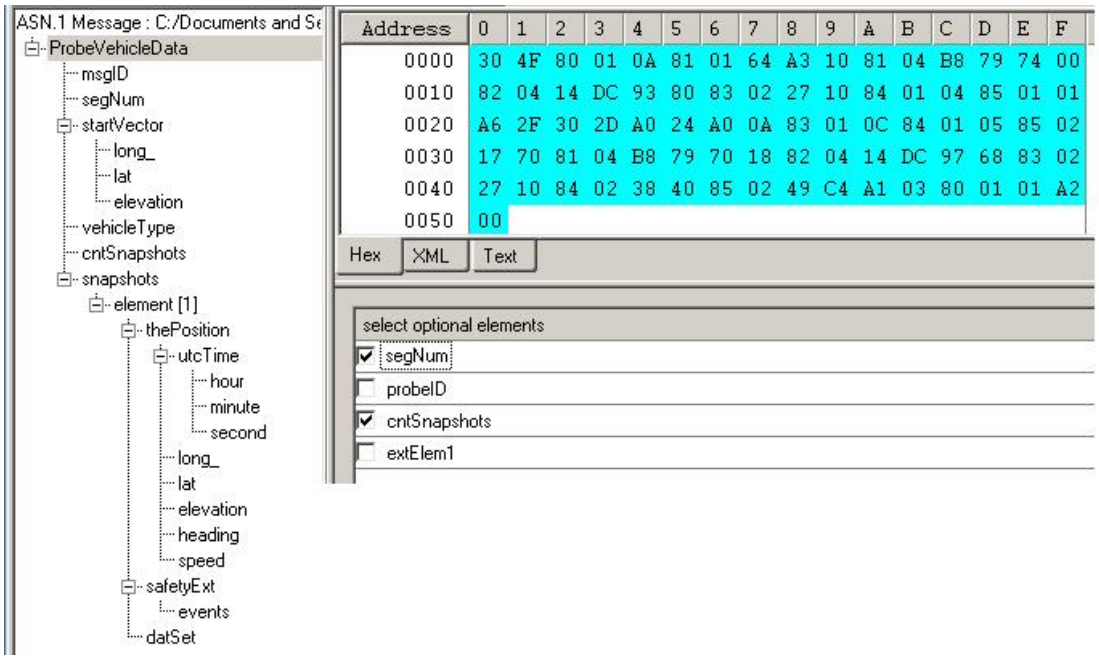

Figure 31 – Typical PVD Message Encoding

This is typical of a valid `ProbeVehicleData` message reporting a single breadcrumb point back to an RSU. It states that there is a single breadcrumb event that occurred at time 12:05:06 at the location -120.0 longitude and +35.0 latitude, at an elevation of 1000 meters above the ellipsoid while the vehicle was traveling due south at a rate of 50 meters per second.

If we were to add a second breadcrumb (with some nearby data, perhaps taken 3 minutes earlier at a point slightly west and north of the other) then the message becomes:

Table 12 – Snapshot Number 2

Element Name	Human Value	Byte Length	Hex Value	Comments
Snapshot # 2				Built up of:
UTC Time				Built up of:
hour	12	4	0x0C	
minute	02	4	0x02	
second	06.000	2	0x1770	
long	-120.0005 deg west	4	0xB8 79 60 78	1/10th micro degree
lat	+35.0005 deg north	4	0x14 DC A7 08	1/10th micro degree
elevation	1000 meters	2	0x03E8	
heading	Due South	2	0x4000	
transmission	forward	-	0x2	Defined by std
speed	25.00 m/s	2	0x04E2 -> 24E2	in 0.02 m/s

This encodes as:

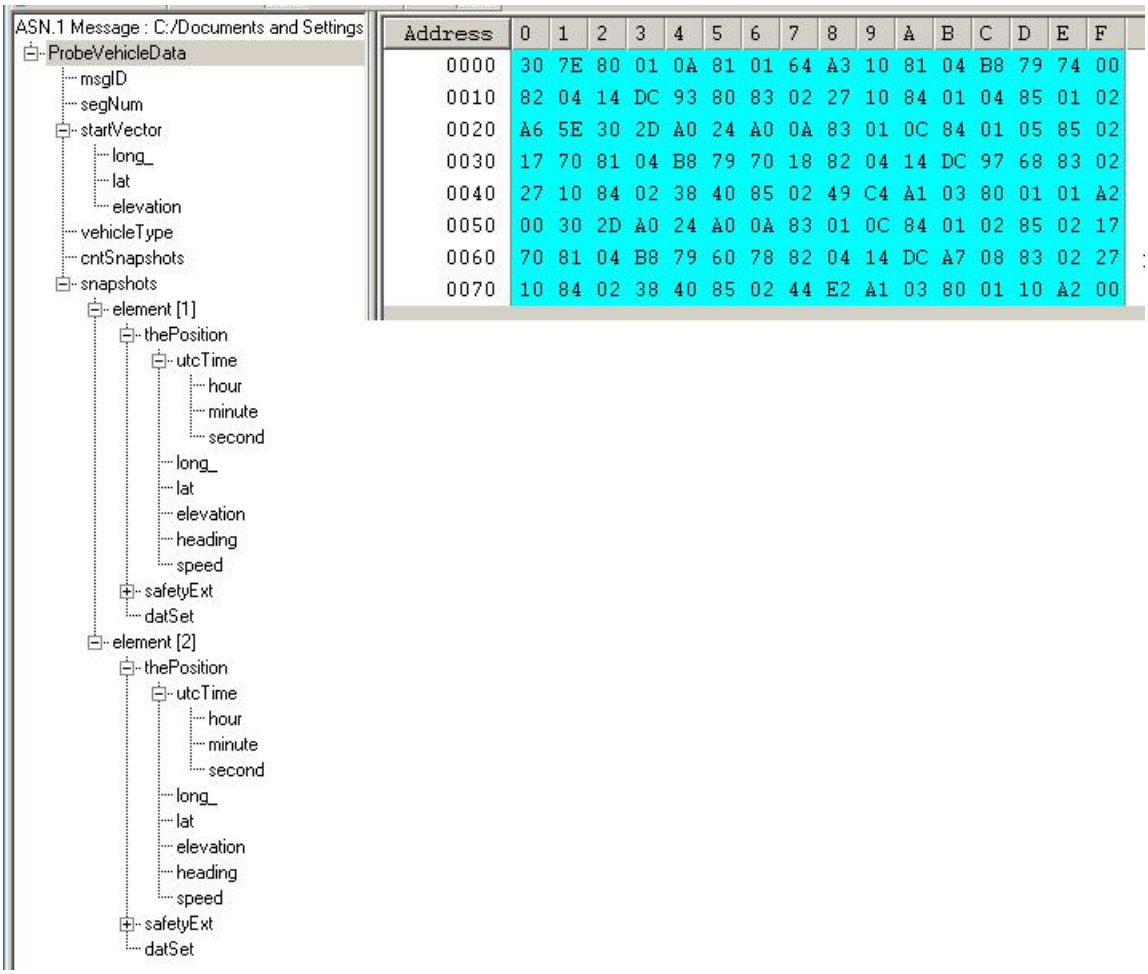

Figure 32 – PVD Message with 2 Snapshots

Note that earlier snapshots or breadcrumbs are transmitted first in order in the list. And the 112 byte length of this message is typical as well.

While only two snapshots are shown in the above, one can add up to 32 total snapshots in a similar way.[68]

5.7 Relevance to ATIS or ITIS Message

The `ProbeVehicleData` message does not use ITIS as a descriptive form of expression (like the `RoadSideAlert` message does). Its use of ITIS codes is limited to a single simple list used in the area of weather rain reporting (one of five items defined in the NTCIP standard and reused by DSRC here). This code (listed below) is no longer a direct copy of the ITIS codes (shown at right) which define their values. No specific qualification is given in the message set standard to determine what rate of moisture relates to each of the defined categories.[69]

[68] Again, subject to any length limits imposed by the lower layers.

[69] Such performance data is likely to be defined in another document dealing with allocating system performance issues for this application.

ITIS type code used in the `ProbeVehicleData` message

`NTCIP.EssPrecipSituation` Grossly Based on `ITIS.Precipitation`

See also the `RainSensor` data element for a simple linear scale of rain rates.

5.8 Interface Design Issues/Considerations

Interface issues here relate to the (possibly quite huge) memory footprint that this application keeps before it dumps to the passing RSU, and how it can routinely run some form of garbage collection algorithm on those data sets to select the final (nominally four) data points that are sent.

5.9 Message Sharing Issues/Considerations

As this is a collection of messages for outbound use, there is in fact little sharing to deal with. Message snapshots would typically be appended into a queue then pruned according to the developed algorithms. This queue would be emptied when a suitable RSU was in view; no sharing of the resulting snapshots would be likely to occur. Exceptions to this might occur if the probe data stream is trying to support some commercial applications, or is providing additional data beyond the minima defined by the standard.

6. Common Issues of ASN.1 Encoding and Decoding

6.1 ASN.1 History and Encoding Styles

The language ASN (or more correctly ASN.1 for *Abstract Syntax Notation revision One*[70]) has been around since the early 1980s. It found its early successes when used for encoding some X.400[71] mail services, then spread to other telecommunications uses thereafter. Today it remains the preferred way to *specify* a data object and is commonly found in that role in many standards.

At its core, ASN is a syntax notation and description system that allows describing data objects in a machine and language independent way. In its earliest forms it was seen as a way for disparate computing *environments* (when large computing machines such as IBM, DEC, Unisys, etc., roamed the earth) and computing *languages* (at that time Fortran, Pascal, PL1, and the new guy "C") to successfully exchange information. ASN allowed expressing an arbitrarily complex data structure of basic building block elements (integers, strings, enumerations, sequences, etc.) in a uniform way for all parties, regardless of how those parties implemented it in their own machines and toolsets. Given this[72] as a basis, one could then encode the defined structure into something that could be transported over heterogeneous networks[73] to another machine which could decode it with similar results.

The *encoding* of ASN in fact splits into several different and distinct styles and forms and has many adherents and detractors for each. It is a constant source of confusion that *ASN syntax* has almost nothing to do with *ASN encoding* – they are two different subjects with the same name. One can (and ITS typically does) use the ASN syntax while totally ignoring any use of any of the ASN encoding choices.[74] We return to this important distinction in a moment.

In the mid 1990s, when the USDOT FHWA ITS program decided to accelerate the development of US standards for ITS use, it was also decided to have the US join in various international standards working groups (for example ISO TC 204). Those groups (and ISO in general) had decided to use ASN as the preferred means of expressing their work, and the US readily agreed to also use *ASN syntax*[75] in its own work. This decision has served the industry well over the past decade and today one can find well written ASN (as well as XML) at the core of most ITS technical standards.

---

[70] We use ASN in this text, although the term ASN.1 is also commonly used.

[71] In fact the ASN of this period was horribly intertwined with the ITU X.400 document, and tended to combine the syntax rules along with the early BER encoding rules in the document. It was not until 1984 (two years later) these were separated out and ASN began to grow on its own and become "the notation-of-choice for the specification of OSI Application Layer Standards" which it remains today. The original work also contains a macro language that by 1998 was recognized as creating problems and ambiguities (it was resolved in that year that no new macros would be written, and the ITS message sets do not in fact use any macros today). You can see some of the macro language still in use among SNMP users and the NTCIP work (in SNMP the existing macros have become *de facto* standards). By the time ITS began using ASN (mid 1990s) the syntax rules were well established and clearly separated from the encoding issues. A good summary of relevant dates in ASN development can be found at: http://www.asn1.org/news.htm#head5

[72] And this small point was fought over for some time (the separation of content definition from its native representation, versus the issues of encoding it to send the content). Here is a rather playful (some might say satirical) link reviewing this period of ASN development history: http://www.btinternet.com/~j.larmouth/tutorials/hist/shortHist.ppt

[73] In this case *heterogeneous* refers to the type of network transport and equipment used. Recall that in the early days even such things as ASCII for text was not uniformly supported among vendors.

[74] The reverse is not typically true. One generally has to have some ASN syntax to then encode into the chosen ASN encoding to be used.

[75] Note *syntax* not *encoding*. There is no official position from the US or any of the ANSI standards developing organizations on the topic of encodings, although specific committees have chosen to recommend specific encodings where there is a need. As an example, the SAE DSRC committee work uses BER-DER encodings; some of the NTCIP committee work uses OER encodings. In general, ITS standards are neutral on this point.

The issue of which *encoding* to use for any given project continues to solicit strong views even today. While most *center to center message* exchanges[76] in ITS today occur using XML (and therefore the XSD schema published by the relevant committee to describe it), other areas of our industry (noticeably device control exchanges defined by the National Transportation Communications for ITS Protocol (NTCIP) and device exchanges in DSRC) still have a legitimate need for a very bandwidth-efficient encoding mechanism.

In such a case, the relevant committee typically selects one or more standardized encoding forms to be used and provides any additional details needed to further control the expressed rendering to ensure interoperability. For example, NTCIP, which is typically used over a Simple Network Management Protocol (SNMP) layer, has chosen[77] to use an ASN encoding style called OER (for Octet Encoding Rules). The strengths and weaknesses of OER are briefly discussed in a subsequent subsection. In a similar fashion, the DSRC committee has selected the Distinguished Encoding Rules (a variant of the Basic Encoding Rules) to be used with its messages when sent over the WSM stack.

As discussed in the remainder of this section, the Basic Encoding Rules provide a simple but robust methodology to encode ASN content for sending. This set of rules is widely implemented in many commercial toolkits, and it is well documented for those that wish to build their own. It has been deployed for over 20 years (with attendant feedback and refinement), and it is commonly seen in such things as the security certificates used on the Internet (as well as for DSRC). It follows the Tag-Length-Value (T-L-V) pattern style of most ASN encoding systems, aligning all values on simple byte boundaries. For the examples developed in this guide we have selected an open source toolkit which implements these rules; the toolkit is covered in greater detail in subsection 6.17 and section 8.

6.2 ASN.1 Resources

The topic of ASN is by nature rather vast and the brief summary provided here does not do it justice. The references listed in subsection 1.5 can be used to find a wide variety of additional source materials. Many educational books on ASN are freely available for download online; consult subsection 1.5 for two of them. ASN is now a mature industry supporting a number of well established vendors of tools, support services, and educational resources to promote their product offerings. And there are off the shelf commercial tools to help decode and understand the bits and bytes over the wire. Subsection 6.18 mentions one such tool, the tool we have used for some of the example encodings which follow.

6.3 Basics of DER Encoding Used in SAE J2735

In the world of data encoding, there is a need to mix tagging (considered wasted bytes) and the actual data (the real payload). Most systems try to keep the overall cost in bytes for the tagging to a minimum; XML by contrast does not.[78] Some encoding systems are expressly designed so that the encoded result can be consumed by the other end without having any sort of specification[79] or schema regarding the content. BER encoding is of this style, and one can readily decode and skip

[76] These are the messages defined by other groups in ITS such as the ATIS message set, which developed the messages used to exchange traveler event advisories. A complete XML schema exists for this message set and this is typically used when it is sent with an XML encoding, as well as service descriptions in WSDL. In the realm of DSRC, XML messages are not sent over the WSM stack but over an IP layer.

[77] In fact this is one of several protocol *profiles* that they have developed. *Normal* SNMP uses BER encoding.

[78] Although this is typically a key concept in XML compression approaches, reducing long tag names to a short token and then applying PKZIP style compression to the remaining ASCII text stream.

[79] The ASN source listing is properly called the *ASN.1 module specification*, however the term *schema* has become very popular since the rise of XML. The terms are treated as equal in this text.

over content in a BER message as demonstrated by the decoding values shown previously in Figure 5 (subsection 3.6). Other approaches depend on the shared schema describing the messages being exchanged beforehand, and can reduce the transmitted content model when this occurs. The style called PER encoding[80] is of this type, as is NTCIP's OER style. Such tricks, reducing the entropy residing in the message, can save transmitted bandwidth at the expense of increased decoding complexity.

What is OER and Where Is It Used?

Practitioners of ITS on occasion come across another encoding system called Octet Encoding Rules and may have questions regarding its application. The OER encoding system was developed by the NTCIP Base Standards and Profiles Working Group to provide a somewhat more effective means of encoding than BER for use in their SNMP device needs. Its use is limited to NTCIP and at this time there are no plans to advance it to become an ITU standard; therefore its market acceptance remains low (there are no longer any known suppliers of toolkits to support OER). The controlling document is endorsed by NEMA*, and is titled *1102 - NTCIP Octet Encoding Rules (OER) Base Protocol* and can be found at: http://www.ntcip.org/library/standards/default.asp?documents=yes&standard=1102

Technically, OER was an attempt to define a better (more compact yet still simple) approach than BER (which does not require an exchange of the ASN specification) by forcing both parties to use and exchange the same specification. It then used this additional shared knowledge of the message structure to be able to drop selected values where a default value was known, in a manner roughly similar to the approach taken by PER. A byte alignment (rather than the bit alignment used in PER) was kept for simplicity.

At this time the NTCIP has backed away from the use of OER over SNMP as a preferred technical solution. Its Datex documents have also been changed, and now reflect the use of PER for the encoding to be used.

* In the three-way relationship between NEMA, AASHTO and ITE that forms NTCIP, NEMA is the only official ANSI certified standards organization. Thus, NTCIP documents derive their legitimacy as US standards from NEMA.

Tagging systems are often described in terms of three key items, the identifying tag used, a length value or type (often in terms of bytes or bits used), and the actual value to be expressed (the int, string, structure, etc.). These values are often referred to by the letters **T**, **L**, and **V**; this naming convention is still used even if the specific item (most often the length) is not explicitly transmitted.

At a gross level encoding approaches can be divided into two basic technical styles. In many systems the tag simply precedes the value (and the length when used); this is often referred as a T-L-V system. This is also referred to as the *definitive length* form. In some systems (such as XML and well-formed HTML) the tag bounds the content with a start tag and an end tag, often expressed as T-V-/T. This is referred to as the *indefinite length* form. As a side effect, this style eliminates the need for the explicit length value in many systems. In such a system, tag nesting may be simple or complex and content may appear alongside tags.[81] In systems which need to encode long content or content where the length may not be known when the tagging process begins (such as streaming video), then the length value is replaced or eliminated and a style of end tag that can be uniquely detected in the content stream is used (some would argue this is simply another T-V-/T style). This inevitably involves additional processing to ensure that the selected end tag pattern does not occur in the data stream values. The end symbol selected may end the object's starting tag (in which case it can be nested) or it may end the processing logic for the message (in which case it typically cannot be). Some systems (the CER variant of BER being one) allow mixing a style of T-L-V use with a style of T-V-/T end tags for longer content items. One can cite many systems using any of these variant methodologies.

[80] Packed Encoding Rules (PER) are a somewhat complex bit-aligned set of rules that takes advantage of known limits and value range defaults to remove as much entropy in the message as possible. In PER, objects are *packed* together using single bits as tag delimiters when possible. A practitioner skilled in data reduction techniques can usually beat the savings offered PER by a slight amount, but at the cost of creating custom rules which depend on additional content knowledge about the source materials. PER is considered *hard* to decode, and fewer tools support using it.

[81] Complex content of this type [i.e., <tag1>content<tag2>content</tag2></tag1>] is allowed in XML but its use is strongly discouraged. In the XML developed in ITS standards, its use is prohibited.

The development of the encoding side of ASN has always had something of split personality in its committee membership, some wanting more complex approaches than others did. The first product of this work (with more advanced products still in progress) was called the *Basic Encoding Rules* and reflects the proponents for a simple approach publishing their work first.[82]

The advent of BER rules produced a couple of unexpected surprises when the market started to use it. The first of these was that, while it did work, the collection of so-called *sender options* allowed multiple interpretations of the standard to create different multiple (and still deemed correct) encodings of the same data set. For example, as developed, the T-L-V style naturally allows nested objects (other T-L-V sets may become the value of an outer set). But BER provided three different ways to encode such a sequence. This was problematic to many people for many different reasons, the need to be able to support and decode each valid variant being one of them. Another critical shortcoming with having different ways to encode the same message content was that security systems using a single validating checksum on the message were unable to cope. The committee went back for round two to address these concerns.

The committee produced two new variants of the BER rules in a single standard:[83] the *CER* encoding (Canonical Encoding Rules) and the *DER* encoding (Distinguished Encoding Rules). Both of these restrict the encoding variants allowed by BER to a *single* set of rules, and thereby produce a single unique set of data. Both produce valid BER encodings. Look up the term *Canonical* to find entries like *"the usual or standard state or manner of something"* or "*from the Arabic word "Qanuun" which essentially means "rule", "law", "standard", and has come to mean "generally accepted" or authoritatively correct"*.[84] This leads to the reasonable question of what *distinguished* means in this context. The best answer is that in fact these are both *Canonical* for their own uses (but the committee needed a second term). The two terms are often confused. But how do they differ? In a nutshell, DER always uses the T-L-V form (the definitive length form) while CER uses the T-V-/T form (the indefinite length form) in some specific places. This makes CER better suited to encoding streams of data such as video or data blobs with unknown lengths. And DER is preferred whenever there is a need for only one specific encoding, such as is needed for many security type uses.

With this further history of encoding out of the way, let us look at the mechanical details of T-L-V encoding produced by BER-DER. These rules are valid for BER as well unless noted. Messages carried about in the datagrams (such as the WSM message in its DER encoded form) are often referred to as Protocol Data Units (*PDUs*) by the layer that carries them. In this context, a PDU is a message (and not a data element in the message or dialog). In BER, the encoding of a PDU (the message) consists simply of cascaded T-L-V encodings, one for each data element defined in the ASN. The encapsulating types supported are `SEQUENCE`, `SET` and `CHOICE`; however `SET` (a sequence of objects in an indeterminate sending order) is never used in ITS work.[85]

In BER, the tags used are based on the characteristics of the *type* of data in the value field. The first three bits of a one byte tag value are used to convey some application scope-related information, while the remaining five bits are used to convey the basic type. In BER literature, the following three tables are frequently used to explain the tag byte encoding (often with the confusing nomenclature that each PDU type is encoded this way[86]).

---

[82] The development of PER was foreseen at that time, but there is no other similar body of work called the *Advanced* encoding rules.

[83] The actual document can be found at: *http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf*

[84] Taken from: *http://www.search.com/reference/Canonical*

[85] Nor is it used in the XML schemas produced in ITS standards work.

[86] This is true for the outermost PDU but not really correct for the inner data elements (which are not strictly PDUs as such because they are always sent as part of the outer one).

Table 13 – BER Tag Encoding Structure, Class Types

8	7	6	5	4	3	2	1
Class		P/C	Number				

Class	bit 8	bit 7
Universal	0	0
Application	0	1
Context-Specific	1	0
Private	1	1

In practical fact only the *Universal* and *Context-Specific* codes are used. And, as will be seen, the Context-Specific tags follow another pattern, that of simply numerically increasing the tag assigned in order (`0x80, 0x81, 0x82,` etc.) with any given level of the ASN specification.

In the above, the P/C bit is used to specify if the value object is a primitive (of a basic type like int) or constructed, meaning that other well-formed T-L-V objects are contained inside of it (i.e., a `CHOICE, SEQUENCE` or a `SEQUENCE OF` statement is being tagged).

The lower 5 bits are used to convey the basic type which is being sent (see the table below). Again this table is only seen when the *Universal* class is being used, therefore it is only typically used for the outermost PDU in a statement.

Table 14 – BER Tag Encoding Structure, Basic Types

Name	P/C	Number (decimal)	Number (hex)
BOOLEAN	P	1	`0x01`
INTEGER	P	2	`0x02`
BIT STRING	P	3	`0x03`
OCTET STRING	P/C	4	`0x04`
NULL	P	5	`0x05`
OBJECT IDENTIFIER	P	6	`0x06`
REAL (float)	P	9	`0x09`
ENUMERATED	P	10	`0x0A`
SEQUENCE and SEQUENCE OF	C	16	`0x30`
SET and SET OF	C	17	`0x31`

Note: `SEQUENCE, SEQUENCE OF` (and `SET`) are of type *constructed* so the Number (hex) is increased by 32 (the P/C bit is set, `0x20`). Notice that `CHOICE` does not appear here, as in practical fact it is treated like a `SEQUENCE` item.

If **Class** is set to Universal, the value is of a type native to ASN.1 (e.g., `INTEGER`). The Application class is only valid for one specific application; in practicality it is not used in our work. Context-Specific depends on the context (and it is the class used for most tags which occur *inside* a sequence). While the private class can be defined in private specifications, it is also not used here.

So, what all this means for a message set like our own which always starts every message with the sequence statement in the form:

```
TypicalMessage   ::=  SEQUENCE {    -- all our messages start with SEQUENCE
   itemA     FirstItem,
   itemB     NextItem,          -- etc.
```

is that the first tag (the first byte of the encoded message) is always 0x30 (a Universal for value and a SEQUENCE for name, along with the P/C bit set to one, hence 0x30).

If we had simple single object messages like the one below, then we would see other initial tags.

```
ErrorCode  ::=   INTEGER(0..10)     -- a one item message
```

In this case the tag would be 0x02 as per the above table (a Universal for value and an INTEGER for name, along with the P/C bit set to zero, hence 0x02). The complete message would consist of three bytes: 0x02, 0x01, 0xXX, where XX is the value, ranging from 0 to 10, to be sent.

The second byte of the encoded message (and potentially the next few bytes if the length was long) is always the length value of the remaining bytes (the bytes used by the value, which may be zero). Hence in the first example of message encoding given in subsection 3.6, the value stream of:

```
0x30 0x31 0x80 0x01 0x02 0x81 0x01 0x01 0x82 0x03 0x00 0xEA 0x60 ...
```

can be decoded to indicate a Universal tag for sequence (0x30), followed by a length value (in this case 0x31), followed by the value (a stream of bytes starting with 0x80). The red boxes shown above denote inner T-L-V sets which are described in a moment. All of the rest of the message is inside the value of the initial sequence element, and its length value reflects this.

Notice the red boxes which are inside the sequence statement do not follow the tag format outlined in the previous table. Rather, they seem to start with 0x80 and increase by one each time. This is correct. It reflects the rules used for objects of the class Context-Specific, which this is.

The Context-Specific class starts *inside* any other sequence object declaration in ASN (and restarts in the case of any nested sequence declarations). Inside the Context-Specific class, objects are given numbered tag assignments in a sequence starting from zero. The upper three bits presented in Table 11 are still used, but the number established by Table 12 is not used, rather only a simple sequential assignment is used. Thus the code:

```
TypicalMessageA  ::=  SEQUENCE {   -- will be tagged as universal tag, 0x30
   itemA     FirstItem,         -- will be tagged as 0x80
   itemB     SecondItem,        -- will be tagged as 0x81
   itemC     ThirdItem,         -- will be tagged as 0x82
   itemD     FourthItem,        -- will be tagged as 0x83
   itemE     FifthItem          -- will be tagged as 0x84
       }
```

results in the tags shown in the comments on the right. These are all Context-Specific type tags. Note that the tag conveys nothing about the underlying base type or what proper defined name the type has or what name that instance of the type has (nor does it ever need to know). In use, each of these is followed by the length and value information in the normal T-L-V form.

This works and is quite simple to use with one exception, based on the P/C bit use. This bit is still asserted when complex content is contained in the value.

When a *native* ASN type is used directly rather than the defined named type, as is common with ITS style ASN, the tagging does not revert to using the Universal class tags. Consider this fragment:

```
TypicalMessageB  ::=  SEQUENCE {   -- will be tagged as universal tag, 0x30
   itemA     FirstItem,         -- will be tagged as 0x80
   itemB     SecondItem,        -- will be tagged as 0x81
   itemC     INTEGER,           -- will be tagged with 0x82 still (no change)
   itemD     FourthItem,        -- will be tagged as 0x83
   itemE     FifthItem          -- will be tagged as 0x84
       }
```

resulting in the same tag shown before for `itemC` (there is no use of a Universal tag of `0x02` indicating an `INTEGER`). The original BER rules would have allowed either tag to be used. The behavior is the same for all other native ASN types in such a context. Note that the other Context-Specific tags remain and are still numbered consecutively. The context numbering includes all the other types present at that level of the code; Universal items do not affect it. It also resets with each new context (typically every new `SEQUENCE`). The scope of this tag assignment is local to the context in which it is used.

One can also expressly define tag values of your own choosing (this is useful when building local ASN that needs to be sure not to conflict with future national standards); here is such a fragment:

```
TypicalMessageC  ::=  SEQUENCE {   -- will be tagged as universal tag, 0x30
   itemA     FirstItem,        -- will be tagged as universal tag, 0x02
   itemB     SecondItem,       -- will be tagged as universal tag, 0x02
   itemC     INTEGER,          -- will be tagged as universal tag, 0x02
   itemD     [10] MyItem,      -- will be tagged as 0x8A  (use of "10")
   itemE     FifthItem         -- will be tagged as universal tag, 0x02
       }
```

Note that the Context-Specific tags numbering has disappeared. By adding the single explicit tag value, all other tags have reverted to Universal tags for everything[87] else, a radically different result! The resulting tag shown for `myItem`, will have a value of 10, and is expressed as `0x8A`, a Context-Specific tag indicating `myItem`. The value *ten* here is defining the explicit Context-Specific tag to be used. Note that the creator of this source code must ensure that the tags selected do not conflict with the machine generated tags and cause ambiguities.[88] If an 11th item (of type Context-Specific) were present at this level, this might conflict. In general, this style is avoided but the ASN library routines will deal with it if the need arises.

When a constructed structure is present (the `SEQUENCE` can nest in such code as shown below for the 4th item), then the assignment of that tag follows the same rules as outlined for the `INTEGER` example above. The tag for this element is taken from the Context-Specific class.

```
TypicalMessageD  ::=  SEQUENCE {   -- will be tagged as universal tag, 0x30
   itemA     FirstItem,        -- will be tagged as 0x80
   itemB     SecondItem,       -- will be tagged as 0x81
   itemC     INTEGER,          -- will be tagged as 0x82
   itemD     SEQUENCE {        -- will be tagged with a 0xA3 (the 4th item)
      itemE     FifthItem,     -- will be tagged as 0x80
      itemF     SixthItem,     -- will be tagged as 0x81
      itemG     SeventhItem    -- will be tagged as 0x82
          }
      }
```

The second `SEQUENCE` is tagged as `0xA3`, indicating a Context-Specific tag of complex content (the P bit is set) resulting in "A", and the "3" continues the consecutive numbering. In this example we see the two most common tags `0x8X` for consecutively numbered simple objects, and `0xAX` for consecutively numbered complex (container) objects. In both cases, the X values (and the first bit of the upper nibble) are determined by the numeric order.

Note that inside the second `SEQUENCE` structure (the `0xA3` type tag) the numbering has a new context (this too is a Context-Specific class) and so it begins numbering the object tags from zero once again.

[87] Note also that the Universal tags are not numbered sequentially like Context-Specific tags are.

[88] And by "dropping back" to use Universal tags for things, the ASN compiler is attempting to avoid this problem. In many use cases where explicit tag numbering is used, every item is numbered to avoid this problem.

If the constructed structure is a defined type from our own ASN then the behavior remains the same. Consider the following:

```
TypicalMessageE  ::=  SEQUENCE {   -- will be tagged as universal tag, 0x30
   itemA     FirstItem,          -- will be tagged as 0x80
   itemB     SecondItem,         -- will be tagged as 0x81
   itemC     INTEGER,            -- will be tagged as 0x81
   itemD     MySeq               -- will be tagged with a 0xA3 (the 4th item)
       }
-- and:
 MySeq ::=  SEQUENCE {
   itemE     FifthItem,             -- will be tagged as 0x80
   itemF     SixthItem,             -- will be tagged as 0x81
   itemG     SeventhItem            -- will be tagged as 0x82
       }
```

Note how `itemD` is treated the same way. There is no additional tagging when the structure is broken up into two parts in this way.

If the above ASN fragment was expressed in an encoded message where all the actual values were set to zero (and therefore lengths were one byte as well), the T-L-V patterns would be as shown in the figure below. In this image, `itemD` is highlighted (and one can see that the length of the enclosed three additional T-L-V items (items E, F, and G) consume 9 bytes).

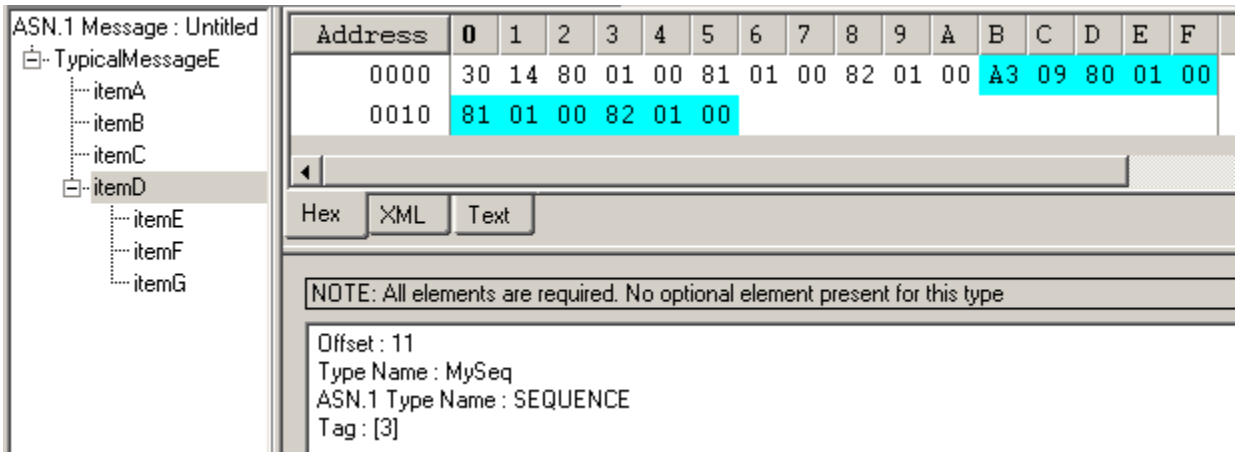

Figure 33 – Encoding of the Message: TypicalMessage

There is one more case of tagging that deserves to be mentioned: structures where more than 32 elements are defined within a single level. While unusual, this is completely legal in ASN and the reader may wonder how this is handled when only 5 bits are reserved for the name index of the tag. Consider a fragment of ASN with many element entries such as:

```
TypicalMessageF  ::=  SEQUENCE {   -- tagged as universal tag, 0x30
   item1     FirstItem,        -- tagged as 0x80
   item2     SecondItem,       -- tagged as 0x81
   item3     MyItem,           -- tagged as 0x82
   item4     MyItem,           -- tagged as 0x83
    --
    -- intervening 25 items dropped from listing
    --
   item29    MyItem,           -- tagged as 0x9C
   item30    MyItem,           -- tagged as 0x9D
   item31    MyItem,           -- tagged as 0x9E
   item32    MyItem,           -- tagged as 0x9F,1F
   item33    MyItem,           -- tagged as 0x9F,20
   item34    MyItem,           -- tagged as 0x9F,21
   item35    MyItem,           -- tagged as 0x9F,22
   item36    MyItem,           -- tagged as 0x9F,23
   item37    MyItem,           -- tagged as 0x9F,24
   item38    MyItem,           -- tagged as 0x9F,25
   item39    MyItem,           -- tagged as 0x9F,26
   item40    MyItem            -- tagged as 0x9F,27
       }
```

The tagging for this begins as one would expect (with values of `0x80, 0x81, 0x82`, etc.) growing to an upper limit of `0x9E` on the 31st item. After that, the tag just uses another byte.[89] It is an understood rule in DER tag decoding that when the space for the number is exceeded (all five bits are ones in this case), another byte is automatically appended. We will see this same extensibility logic used when we address value encodings for integers and for lengths. Note that when the new byte is used in the tag, the value is not reset[90] (it is `0x1F` not `0x00`). Additional tags simply continue with this new pattern as needed.

The above shows both the *Universal* style and the *Context-Specific* style of tags which are used in the DER message encoding format. The discussion here has concentrated on the selection of the tag byte used in the T-L-V system. In subsequent subsections we consider how the length and value bytes are formed, a topic which varies slightly based on the type of data being encoded.

6.4 Value Encodings In General

Each of the subsections to follow describe the basic issues associated with that data type, and give representative examples of encoded values. For example, an integer can be from 3 to 6 bytes of payload size depending on its range (the first two bytes being the tag and length). Many of the integer values used in DSRC tend to be 0...127 (1 byte) but may, on statistically remote occasions, be much larger. The encoding method offered by DER takes advantage of this to use as few bytes as possible to represent the data value.

The message `TypicalMessageG` found in the supporting text file *Section6_ASN_examples.txt* can be helpful here using an ASN viewer tool. It contains one instance of each basic type with no size or length restrictions, and can be used by ASN encoding viewer tools to create simple messages with the content under discussion.

One key concept that cuts across all value ranges for almost all types of data is that of sign extensibility. In a nutshell, if the value being sent could be confused as to sign used (i.e., does hex value `0xFF` represent the value 255 or signed -1?) then an full extra byte is used to extend and convey the sign bit extension.

[89] The extensibility process would repeat again with the 127th item assignment to become a 3 byte tag which could hold the next ~32k tags before overflow to a 4th byte would occur.

[90] The possible values in the 2nd byte field for tags 0x00 to 0x1E are in fact discarded and not used.

This technique is also used in the length bytes (which are considered to be unsigned). However, as DSRC messages are very short (and therefore the length rarely exceeds 127 bytes[91]) it is not often seen.

For each basic type, a few representative typical encodings are shown with each T-L-V in a small red box. The tag values used for this follow the Context-Specific class of use, which is what is generally found in the DSRC message set when encoded in DER. Numbering of the tags are simply 0,1,2,3 etc., to represent the assigned tag values in a fictitious SEQUENCE container.

Sign Extensibility in ASN Encoding

Many ASN values use the concept of *Sign Extensibility* to preserve the sign of the transmitted value. As a result, there really is no concept of an unsigned integer in many places. In these cases, the most significant bit which may contain the sign is always preserved and can be extended into the bits of another additional byte for the value. The BER encoding provided by the ITU standard states the rule in this way:

Clause 8.3.2 If the contents octets of an integer value encoding consist of more than one octet, then the bits of the first octet and bit 8 of the second octet:
a) shall not all be ones; and
b) shall not all be zero.

Note: These rules ensure that an integer value is always encoded in the smallest possible number of octets.

Clause 8.3.3 The contents octets shall be a two's complement binary number equal to the integer value, and consisting of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn, up to and including the last octet of the contents octets.

Note: The value of a two's complement binary number is derived by numbering the bits in the contents octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 of the first octet. Each bit is assigned a numerical value of 2N, where N is its position in the above numbering sequence. The value of the two's complement binary number is obtained by summing the numerical values assigned to each bit for those bits which are set to one, excluding bit 8 of the first octet, and then reducing this value by the numerical value assigned to bit 8 of the first octet if that bit is set to one.

6.5 Encoding Integers of Various Lengths

Integers are always transmitted with the smallest number of bytes needed to represent them (including a sign value), regardless of what range restrictions may be used in the ASN specification. Therefore a value defined as `INTEGER (0..65535)` but which happened to be set to 10, will occupy one byte for its transmitted value. By contrast, one set to 65535 will occupy three complete bytes (`0x00`, FF ,FF) due to the need to preserve the sign value. If the design calls for conservation of bytes (as DSRC does) the effective value range which can be fitted into each byte is half the unsigned range.

A sequence of simple encoded integers looks like:

Given 1,2,3,4 it becomes: `0x 80 01 01, 81 01 02, 82 01 03, 82 01 04`

As mentioned above, integers follow the extensibility rule: the transmitted upper bits must always carry the sign bits and extra value bytes are added as needed to achieve this. In DER encoding, there really is no such thing as an unsigned integer range.

Thus:

Given -1,-2,-3,-4 it becomes: `0x 80 01 FF, 81 01 FE, 82 01 FD, 82 01 FC`
Given 126,127,128,129 it becomes: `0x 80 01 7E, 81 01 7F, 82 02 00 80, 82 02 00 81`

Note the extra byte used in the last two values. Similar rollover points occur at 128, 32767 and -32768.

6.6 Encoding Enumerated Values

Enumerated values are treated as a special case of integers, and they are presumed to be signed.

Enumerated values are always transmitted with the smallest number of bytes needed to represent them (including a sign value), regardless of what range of values may be used in the ASN specification. Therefore a value defined as 10 occupies one byte for its transmitted value as does a

[91] Ed: Because it is unsigned.

value of 127 or -127. By contrast, one set to 128 occupies two complete bytes (0x00, 80) due to the need to preserve the sign value. Therefore the range of enumerated values reserved for local use in most ITS work (128 to 255) will occupy two or three bytes when sent. The names assigned to the values in an enumeration will show up in the tool and library used[92]; they are not used for the over the air values in the T-L-V in any way.

A sequence of simple encoded enumerated values looks like:

Given 1,2,3,4 it becomes: 0x 80 01 01, 81 01 02, 82 01 03, 82 01 04

As mentioned above, enumerated values follow the extensibility rule, the transmitted upper bits must always carry the sign bits and extra value bytes are added as needed to achieve this.

Thus:

Given -1,-2,-3,-4 it becomes: 0x 80 01 FF, 81 01 FE, 82 01 FD, 82 01 FC
Given 126,127,128,255 it becomes: 0x 80 01 7E, 81 01 7F, 82 02 00 80, 82 02 00 FF

Note the extra bytes used in the last two values. Similar rollover points occur at 128, 32767 and -32768.

If the ASN specification does not assign explicit values to the enumerated strings, then the ASN tool is likely to do this for the user, typically beginning with zero and incrementing with each item.

6.7 Encoding Character Strings of Various Lengths

Character strings (IA5String types[93]) are always transmitted with the smallest number of bytes needed to represent them (there is no sign value), regardless of what range restrictions may be used in the ASN specification.[94] These rules are also followed for the derived types such as PrintableString or NumericString as well.

Therefore a value defined as IA5String (SIZE(0..5)) but which happened to set to a value of "ABC", occupies 3 bytes for its transmitted value. By contrast, if the data element is one defined as IA5String (SIZE(5)) that same value would be padded by the encoded logic to produce the needed 5 bytes. The ASN library would produce a value string of "ABC " due to the ASN rule of *big-endian* transmission. Note that the two spaces in this example would each be sent as a value of 0x00, not as an ASCII space (0x20). The complete ASCII character set can be sent, but there is no need to send any sort of end of string delimiter ('\n') in this system. If required in the programming language at either end, it can be added there.[95] Note that prohibited characters can be sent in such a system, until the ASN specification is used to validate the content.[96]

Some encoded IA5Strings look like:

Given “ABCDEFGHIJ” it becomes: 0x 80 0A 41 42 43 44 45 46 47 48 49 4A
Given “” it becomes: 0x 80 00
Given “Hello World” it becomes: 0x 80 0B 48 65 6C 6C 6F 20 57 6F 72 6C 64

[92] They are often mangled in some way to fit the language restrictions.

[93] The IA5 part of this name stands for International Alphabet number five, the ISO term for common ASCII. In DSRC we typically do not use the UTF8String or the BMPString types. Each of the many string types has its own tag value in the Universal class of tags. Here is a more complete list than that in the table given before, which includes these derived char types: *http://www.obj-sys.com/asn1tutorial/node124.html*

[94] However, note that some ASN specifies a fixed length and then directs the user (not the ASN library) as to how to stuff data into that length, such as padding with leading spaces as in " ABC" for a five character string. Such a value would be sent as 5 bytes.

[95] Some ASN libraries do this automatically, but most use fixed arrays for strings.

[96] ASN is similar to XML in this regard; one can legally encode value content in the tagging and send it about, as long as it is *well-formed*. Until that content is checked against the schema restrictions, its illegal nature is not detected.

6.8 Encoding Bit Strings and Named Bit Strings Values

An ASN `BIT STRING` type is always transmitted with the smallest number of bytes (not bits) needed to represent it (there is no sign value), regardless of what range restrictions may be used in the ASN specification.[97] However, at least one byte is always used (there is never a zero length `BIT STRING` value), as noted below. Because the BER-DER form is always byte aligned, padding occurs to make the number of transmitted bits align with a byte boundary (multiples of eight). In the value created, the eighth bit is always used first, with other bits in each byte used thereafter. Zeros are always sent, they are not truncated.

Frankly, the encoding of the `BIT STRING` type is the hardest of the DER process rules to understand. The odd encoding of the first (initial) value byte is uniquely different from the rest of the simpler value encodings, where only the sign extensibility rules gives any complexity. This first byte is used to tell how many bits are in fact used in the *last* byte of the rest of the value. Quoting from the relevant ITU standard itself, we have the following rules:

> *Clause 8.6.2* The contents octets for the primitive encoding shall contain an initial octet followed by zero, one or more subsequent octets.
>
> *Clause 8.6.2.1* The bits in the bitstring value, commencing with the leading bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first subsequent octet, followed by bits 8 to 1 of the second subsequent octet, followed by bits 8 to 1 of each octet in turn, followed by as many bits as are needed of the final subsequent octet, commencing with bit 8.
>
> **Note**: The terms "leading bit" and "trailing bit" are defined in ITU-T Rec. X.680 | ISO/IEC 8824-1, 21.2.
>
> *Clause 8.6.2.2* The initial octet shall encode, as an unsigned binary integer with bit 1 as the least significant bit (LSB), the number of unused bits in the final subsequent octet. The number shall be in the range zero to seven.
>
> *Clause 8.6.2.3* If the bitstring is empty, there shall be no subsequent octets, and the initial octet shall be zero.

In ASN, a `BIT STRING` is defined as *binary data of arbitrary length* (i.e., the value field is an ordered sequence of zero or more bits). Because at least an entire byte is used in transmission, bit strings developed in the DSRC standard (and for most of ITS for that matter) often tend to have 7 or 8 items to try and take advantage of byte boundaries.

However, when we used named bits (as is often the case in DSRC), some savings can occur if we place the less likely to be used bit fields higher in the set of named bits. By doing this, the resulting value can be treated somewhat like a smaller integer (saving bytes in transmission), and like a larger integer only when those bits are asserted (set to one). The ASN fragment called `TypicalMessageJ` can be used to illustrate this behavior.

Some ASN specifications achieve this same effect by defining sets of enumerated bits as integers (rather than a bit string) and then simply defining the bit meanings and relationships outside of the ASN environment. A use of this technique can be found in the 2 bit DSRC definitions for many safety systems,[98] which are in turn combined to make byte long composite values. In such a case[99] the encoding rules are those outlined for integers.

---

[97] However, note that some ASN specifies a fixed length and then directs the user (not the ASN library) to stuff data into that length.

[98] These safety systems are typically defined to include “not present” as well as on/off/activated, hence 4 states.

[99] Caution should be used when doing this. Note that if the most significant bit is ever asserted, the byte will be treated as a signed value and in fact 2 bytes will be transmitted, causing unintended waste.

An encoded `BIT STRING` value looks like:

Given “10000”	it becomes: `0x`	`80 02 03 80`	-- the 03 indicates 3 unused bits
Given “1000”	it becomes: `0x`	`80 02 04 80`	-- 04 indicates unused bits, etc.
Given “100”	it becomes: `0x`	`80 02 05 80`	
Given “10”	it becomes: `0x`	`80 02 06 80`	
Given “0”	it becomes: `0x`	`80 02 07 00`	
Given “1”	it becomes: `0x`	`80 02 07 80`	
Given “01”	it becomes: `0x`	`80 02 06 40`	
Given “001”	it becomes: `0x`	`80 02 05 20`	
Given “0001”	it becomes: `0x`	`80 02 04 10`	
Given “00001”	it becomes: `0x`	`80 02 03 08`	
Given “000001”	it becomes: `0x`	`80 02 03 04`	
Given “0000001”	it becomes: `0x`	`80 02 01 02`	
Given “00000001”	it becomes: `0x`	`80 02 03 01`	
Given “000000001”	it becomes: `0x`	`80 03 07 00 80`	
Given “100000000”	it becomes: `0x`	`80 03 07 80 00`	
Given “10101”	it becomes: `0x`	`80 02 03 A8`	
Given “01010”	it becomes: `0x`	`80 02 03 50`	
Given “00000”	it becomes: `0x`	`80 02 03 00`	
Given “11111”	it becomes: `0x`	`80 02 03 F8`	
Given “ “	it becomes: `0x`	`80 01 00`	-- A null value sent

6.9 Encoding Boolean Values

Booleans are always transmitted with one byte used to represent their value. This is rather wasteful and it is why most Booleans (or any bit fields smaller than a byte) found in the DSRC work have been combined into sets of data which take up one or two bytes (and which are then sent as a value with the joined content inside of it).[100]

The value sent is either `0x00` for false or `0xFF` for true.

An encoded Boolean value looks like:

Given TRUE	it becomes: `0x`	`80 01 FF`
Given FALSE	it becomes: `0x`	`80 01 00`

In the original BER, this value could be encoded many ways (only the sign bit of the value was tested) but in the development of DER and CER the allowed options have been reduced to the above single values.

6.10 Encoding Object Identifier Values

The whole topic of Object Identifier (OID) values is not much used outside of the SNMP protocol environment. It is not used in DSRC at all. However, it is considered a primitive type in ASN, so we briefly cover it.

A well-formed OID consists of one or more dot separated values (each an integer with only a positive range). The maximum number of bytes is not specified, but rarely exceeds 8 or so. Some groups divide the ranges so that the individual values never exceed 255 (an unsigned byte), others do not. So an OID fragment of 0_1_2_45160_10_128 is considered valid. Such a coding is

[100] Like the data blob as an octet case, such inner encodings are not known to the ASN layer and require the application layer to deal with decoding them.

expressed as a stream of signed bytes. Any leading zero values are dropped. Any large values are sent using the sign extensibility rules.

An OID looks like:

Given 0-1-2-45160-10-128 it becomes: `0x 80 06 01 02 E0 68 0A 81 00`

Note that the value 45160 has become `0xE0` 68, while the value 128 has become `0x81 00`.

Note also that here is another case where we see the universal rule of ASN encoding that the larger (more significant) bits are always transmitted first (the *big-endian* rule) and some sign extensibility is again used. This is made more complex due to some limits regarding how many nodes the early part of an OID can contain, that issue can cause encoding problems.

Quoting from the ITU standard one more time:

> *Clause 8.19.2* The contents octets shall be an (ordered) list of encodings of subidentifiers (see 8.19.3 and 8.19.4) concatenated together.
>
> Each subidentifier is represented as a series of (one or more) octets. Bit 8 of each octet indicates whether it is the last in the series: bit 8 of the last octet is zero; bit 8 of each preceding octet is one. Bits 7 to 1 of the octets in the series collectively encode the subidentifier. Conceptually, these groups of bits are concatenated to form an unsigned binary number whose most significant bit (MSB) is bit 7 of the first octet and whose least significant bit is bit 1 of the last octet. The subidentifier shall be encoded in the fewest possible octets, that is, the leading octet of the subidentifier shall not have the value 8016.
>
> *Clause 8.19.3* The number of subidentifiers (N) shall be one less than the number of object identifier components in the object identifier value being encoded.
>
> *Clause 8.19.4* The numerical value of the first subidentifier is derived from the values of the first two object identifier components in the object identifier value being encoded, using the formula:
>
> (X*40) + Y
>
> where X is the value of the first object identifier component and Y is the value of the second object identifier component.
>
> **Note**: This packing of the first two object identifier components recognizes that only three values are allocated from the root node, and at most 39 subsequent values from nodes reached by X = 0 and X = 1.
>
> *Clause 8.19.5* The numerical value of the i^{th} subidentifier, ($2 \le i \le N$) is that of the $(i + 1)^{th}$ object identifier component.

6.11 Encoding Data Blobs (Octet Sequences)

Octets are always transmitted with the smallest number of bytes needed to represent them (there is no sign value), regardless of what range restrictions may be used in the ASN specification.

Therefore a value defined as `OCTET STRING (SIZE(0..5))` but which happened to set to a value of `0x01, 02, 03`, will occupy 3 bytes for its transmitted value. By contrast, if the data element is one defined as `OCTET STRING(SIZE(5))` that same value would not be padded by the encoded logic to produce the needed 5 bytes. Rather, the two last "zero" bytes are dropped in favor of sending less bytes. The ASN library would produce a value string of `0x01, 02, 03`, which meets the ASN rule of *big-endian* transmission as well as that of being byte aligned.

An octet value looks like:

Given “5” it becomes: 0x 80 01 50 Note that “0x50” is sent
Given “05” it becomes: 0x 80 01 05
Given “00 11 22 33” it becomes: 0x 80 04 00 11 22 33
Given “AC 1D” it becomes: 0x 80 02 AC 1D
Given “DE AD” it becomes: 0x 80 02 DE AD
Given “FF DD EE AA DD” it becomes: 0x 80 05 FF DD EE AA DD

As always with octet blobs, any inner meaning (implicit encoding) that the blob may contain is hidden from the ASN encoding layer. Any further decoding or checking becomes the responsibility of the application layer.

Note also that the first example (sending 0x5) shows another case where we see the universal rule of ASN encoding that the larger (more significant) bits are always transmitted first (the *big-endian* rule). In this case the data is stuffed with a lower nibble of “0” to make 0x50 to cause the transmitted data to occupy a byte aligned amount of space.[101] Note that this does not occur for the other examples, as they are already byte aligned. Additional empty bytes are never padded to the end of the value either, but a value with 0x00 could be transmitted.

6.12 Encoding a SEQUENCE or CHOICE Structure

In the Context-Specific areas inside a message, the treatment of enclosing structures like SEQUENCE, SEQUENCE OF, or CHOICE is all the same. They are given a tag with an assigned numeric value (and with the P/C bit set) so the resulting hexadecimal tags are seen to be values like 0xA0, A1, A2, A3, etc.

Consider the ASN fragment below:

```
TypicalMessageI  ::=  SEQUENCE {  -- tagged as universal tag, 0x30

  aSEQ        SEQUENCE {          -- tagged as 0xA0
   item1      FirstItem,          -- tagged as 0x80
   item2      BOOLEAN,            -- tagged as 0x81
   item3      INTEGER,            -- etc. as before.
   item4      BIT STRING,         --
   item5      OCTET STRING        --
        },

  aSEQ-Of SEQUENCE OF MyItem,     -- tagged as 0xA1
                                     -- inner items of MyItem
                                     -- are tagged with 0x80, 81, 82 etc.

  aCHOICE  CHOICE {                  -- tagged as 0xA2
   item1      FirstItem,          -- tagged as 0x80
   item2      BOOLEAN,            -- tagged as 0x81
   item3      INTEGER,            -- tagged as 0x82
   item4      BIT STRING,         -- tagged as 0x83
   item5      OCTET STRING        -- tagged as 0x84
        }
  }
```

This shows all three types of structures in their typical usage. The resulting tags are shown in comments on the right. Note that each of these establishes a new context for the elements found within it (and therefore the numbering starts again from zero for items at that level).

[101] In fact, some would claim that ASN really does not have an octet as a primitive type; it uses the phrase “…binary data whose length is a multiple of eight…” as its basic definition.

For a `SEQUENCE` inside of the message and the `SEQUENCE OF`[102] inside of the message, the form of the encoding is simple, a T-L-V (such as `0xA0`, length, value stream).

For a `CHOICE` inside of the message, note that only one item can be transmitted inside it; however the tagging assignment to the possible items follows the same logic as the other cases.

6.13 Dealing with OPTIONAL Content Elements

This issue actually does not come up in selecting tags and values for encoding. If an element is not to be sent, it is simply skipped (no T-L-V is present). The presence or absence of an element does not affect the tag numbering used for that structure (as all possible items are numbered). If one sees tag numbers which skip in value in the received ASN, one can infer that there was an optional element not sent.

This issue does have to be addressed in the actual memory structure used to hold such a message, so that one can determine if an optional item was present in a transmitted message. This is discussed in a moment when we consider issues with how the ASN is represented in memory.

One odd occurrence that does happen on occasion is an empty data value structure (that is a value which is nothing). This can occur when the value element (or elements, it happens most frequently in sequences[103] with no content) provides no value to be sent. In such a case, the length is simply set to zero (the T-L-V then becomes a 2 byte value of `0x8x-00` or `0xAx-00`, etc.).

6.14 Issues with ASN Held in Memory

Before and after the encoded ASN message (or more correctly, its encoded payload bytes) is transmitted, it is kept in the application memory space. The precise form that the message structure takes is determined by the ASN tool that is used. ASN tools vary considerably on this point, but in no event is there a strict *one to one* mapping between the message structure and the memory copy of the message. Various additional items (metadata) are required and nested structures and sequence are often handled by way of pointers.

Consider a fragment of ASN as made up of four bytes of data as follows, and encoded using the normal BER-DER methods to have a T-L-V for each element in turn.

```
ExampleMessage ::= SEQUENCE  {              -- tag 0x30, length 0x0C
   itemA  INTEGER (0..31),                  -- tag 0x80, length 0x01
   itemB  INTEGER (0..100),                 -- tag 0x81, length 0x01
   itemC  INTEGER (0..100) OPTIONAL,        -- tag 0x82, length 0x01
   itemD  INTEGER (0..100)                  -- tag 0x83, length 0x01
   }
```

In receiving this message, each inner item is constructed in exactly three bytes in total. The T-L-V triplet consists of a first byte for the tag, indicating items A,B,C,D as being the 1st, 2nd, 3rd, 4th items at that level (there is no indication that they are bounded integers), then a length value (in this case always one byte in length[104]), and then the actual values of A, B, C, D, in turn. After completely decoding `itemB`, the next item will be *either* `itemC` or `itemD`, and the tag found indicates which it is

[102] The `SEQUENCE OF` keyword is used to specify a sequence of the same single element.

[103] A sequence which contains nothing but optional elements but which requires the sequence itself to be always sent is the most typical cause of this. It is typically an improvement to the ASN to make the sequence optional as well, in such cases, and will save an additional two bytes of message payload.

[104] Note that the length is the number of whole bytes needed to hold the current instance of the value. In this case, even the largest values (32 or 100) of the items still requires one byte to hold. In BER and DER, objects are counted in even bytes; in other encoding systems (notably PER) some savings may occur because objects are counted in bits, so a range like 0..31 or even 30..61 can be fitted into 5 bits. Note that the sign extension of the values is still used here.

(either 0x82 or 0x83). At first blush one might presume the memory footprint to contain this structure would simply be as follows:

```
typedef struct ExampleMessage {      typedef struct ExampleMessage {
      int       itemA;                       BYTE   itemA;
      int       itemB;     or perhaps even:  BYTE   itemB;
      int       itemC;                       BYTE   itemC;
      int       itemD;                       BYTE   itemD;
      } ExampleMessage_t;                       } ExampleMessage_t;
```

However, we have this OPTIONAL keyword to deal with. As is often the case, there is no spare room in the above memory structure to hold the meta information regarding the presence or absence of the data in the data itself. In other words, in the *.h fragments above, one cannot tell by inspection the difference between an itemC set to a value of zero and an itemC that was in fact absent in the transmitted message.

One might decide to encode this information in a part of one of the integers that does not run the full range (0..255) or in another supporting structure, but the ASN committee wisely decided against such an encoding (making the resulting rules more scalable to any content types). What is then needed is an additional *inline* data element that can be used to determine the presence or absence of itemC. This item is used in the *memory footprint*, and by the encode-decode logic to build messages, but it is never transmitted. The ASN specification does not require all implementations to solve this in the same way, and for this reason code libraries produced by one ASN.1 tool cannot easily be used on another tool.[105] A simple way to solve this particular need is to add an additional Boolean type value to the memory structure which can be tested to determine the presence or absence of the element, as follows:

```
typedef struct ExampleMessage  {
      int       itemA;
      int       itemB;
      bool      isItemCPresent;
      int       itemC;
      int       itemD;
      } ExampleMessage_t;
```

The above structure can then be passed about in memory between the application level code and the ASN.1 library code. Typically (this again varies with the tool used) the Boolean value is never directly modified by the application software, leaving metadata management to the library code (which often is built up of small Set()-Get() calls which manage such details).

If the structure uses pointers to represent the parts of other data structures, then it must point to a valid structure, if that structure is required in the messages. Pointers are required to handle indeterminate list or array type values, but this is also commonly used for any nested structures. If the pointed-to structure could itself in fact be empty, then it can be populated with null values as well. If the object pointed to by the pointer can be optional, it is common to use a NULL pointer to represent that none will be sent. This approach works well with nested structures of all types.

However, this technique can be taken to extremes as a mechanism for optional content as well. Consider the approach of representing *any* object which is declared optional with a pointer in the primary memory structure for that message. Then, if the pointer is set to NULL, the object was not present. The ASN1c tool which we have chosen to use takes this approach. Here is the actual memory footprint for the optional example produced by the ASN1c tool:

[105] This is not the only difference. Another key point is that some tools have extensive element range checking while other tools have little or none, and still others allow controlling the level of code checking in the generated libraries. This area is frequently a critical difference depending on the performance requirements of the user.

```
/* ExampleMessage */
typedef struct ExampleMessage {
    long    itemA;
    long    itemB;
    long   *itemC    /* OPTIONAL */;
    long    itemD;
    /* Context for parsing across buffer boundaries */
    asn_struct_ctx_t _asn_ctx;
} ExampleMessage_t;
```

Note that the simple byte sized values have been represented by `longs` (perhaps a bit wasteful, but certainly generic in approach[106]). The 3rd item is however represented by a pointer to a `long`. If this pointer is set to `NULL`, then it indicates that no third item is (or was) present.

6.15 Putting It All Together in Messages

In short summary then, the use of DER encoding rules in the style we have used always involves an outermost tag indicating sequence (`0x30`) followed by a length value (typically a single byte), then sequences of T-L-V pairs which can each be decoded in turn. Each of the subsequent tags, at each new level of the ASN object, is numbered from zero onwards (resulting in tags of `0x80`, `0x81`, `0x82`, etc.). If a tag contains any nested complex content, then its P/C bit is set, resulting in the "8" becoming an "A" as in the tag `0xA0`. Nesting restarts the tag numbering at each new level (a `SEQUENCE`, `SEQUENCE OF` or `CHOICE` statement causes this). The length byte represents the minimum number of bytes needed to contain the value that follows (and can be zero). The actual value field includes provisions for *extensibility*, in that for some types of data the sign bit is always preserved and this may cost an extra byte at times. Data is always sent *big-endian* and byte aligned; bits will be added if needed. Items marked as `OPTIONAL` are still numbered in the tag assignment process, then can be detected when sent by the tag used. Note that the decoding process can be done without having the original ASN source specification at hand, although if it is present then meaningful names can be assigned to the recovered elements. All of this can be implemented by the developer, or a library can be used to provide the functionality.

The final package of encoded bytes is then sent to the next layer for transmission. Typically in DSRC this is handed over to the IEEE1609 layer where an IP style datagram is created. Reception involves the opposite flow of events. The most common means of handling this is to place the ASN library directly in line with the sending or receiving socket communications as was shown in Figure 3 subsection 3.5.

6.16 Products of the ASN.1 Tool

The need for an ASN tool occurs because the alternative is to write application code embodying all the encoding rules which we have covered (and more) in this section.[107] An ASN library, created to match the names and types of ASN objects to be found in our own messages, allows us to handle the encoding details more effectively. The ASN library itself is created with an ASN tool which is typically acquired from a third party. While the specific details and usage conventions of these tools vary, they all share some basic functionality. This subsection describes how such tools are used and how they fit into the application's development process flow.

[106] And given what the reader now knows about encoding integer value types, one can see that this allows the tool to transparently deal with the size of the value bytes to be used.

[107] This is simply another of life's *buy vs. build* acquisition decisions, but most developers conclude it is better to acquire such a library than to develop it in-house.

The primary purpose of the ASN tool is to produce these libraries. Its use is primarily in the early stages of the program lifecycle, when the ASN may be changing and being refined. Every revision of the ASN source specification causes a loop through the tool to reproduce the library files for that baseline. Once the source of ASN is stable (as would be expected in a published standard) then the effort is typically reduced to simply deleting the ASN portions not used,[108] and running it into the tool.

The ASN tool itself is simply a compiler which reads and parses ASN and then generates relevant output files. The resulting pile of files (invariably there are many files produced) can then be used as a whole, generically referred to here as the *ASN library*. This process flow is illustrated below. It is typically repeated whenever there are changes to the ASN source specification.

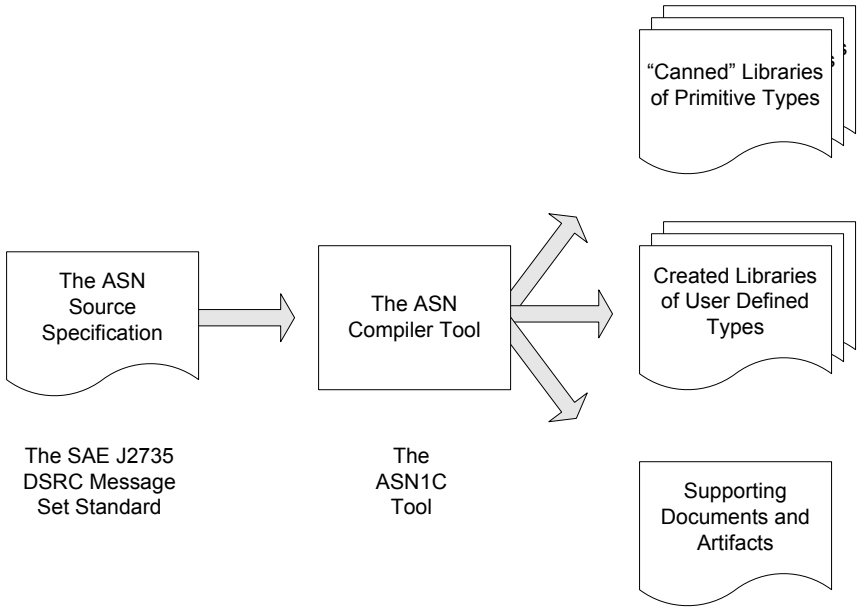

Figure 34 – ASN Tool Use and Products

The library files which are produced are source code listings for the target language which the tool supports. They must then be complied and linked with other files to produce a composite application, using whatever development environment the user chooses.

For our DSRC guide examples, we have produced libraries in simple C and will use these in a C++ application developed in the Windows Design Studio environment to produce a Windows style application.[109] Thus our design language is C/C++ and our library files consist of *.c and *.h files. Other languages are supported in other ASN tools, but C and Java remain the most common.

The files placed into the above library tend to fall into two patterns, those of generic or *canned* support files and those of *user defined* support files. The user support files represent the defined type names of objects in the ASN specification (such as `msgType`) while the generics support the fundamental objects found in the ASN language itself (such an as `ENUMERATED` object). The

[108] One could use the entire ASN specification as issued, but this creates additional files (clutter) with which to deal. If every module of code needs to be controlled and tested in a development environment, this may be a sufficient reason to remove unused items. Most compilers will drop extraneous code that is never used in the linking stage, so the choice of editing the files versus leaving them intact becomes one of personal preference.

[109] Further details of this are provided in section 8. We have in general tried to separate the "pure DSRC" and "pure ASN" sections of the code into their own modules. This was done to separate the guide examples from the Windows framework so that users can more easily use the code in embedded development environments of their own.

generic files do not change based on the user's ASN, they are simply included in the output if that basic type is used. By contrast, the created libraries are typically simple template documents where the named type (taken from the user's ASN specification) is dropped into the appropriate template for the basic type that makes it up. These created files in turn call functions in the generic files to implement the functionality that is needed. This process is repeated for all objects defined in the ASN specification, including the top level messages.

The end result is a lot of files, typically two per generic and two per object defined in your ASN specification (a C and an H file in each case). Consider the list of library files shown below. This comes from the very first (and therefore simplest) example ASN specification containing only a shortened version of the BSM message. This brief ASN produced 72 files in the resulting library.

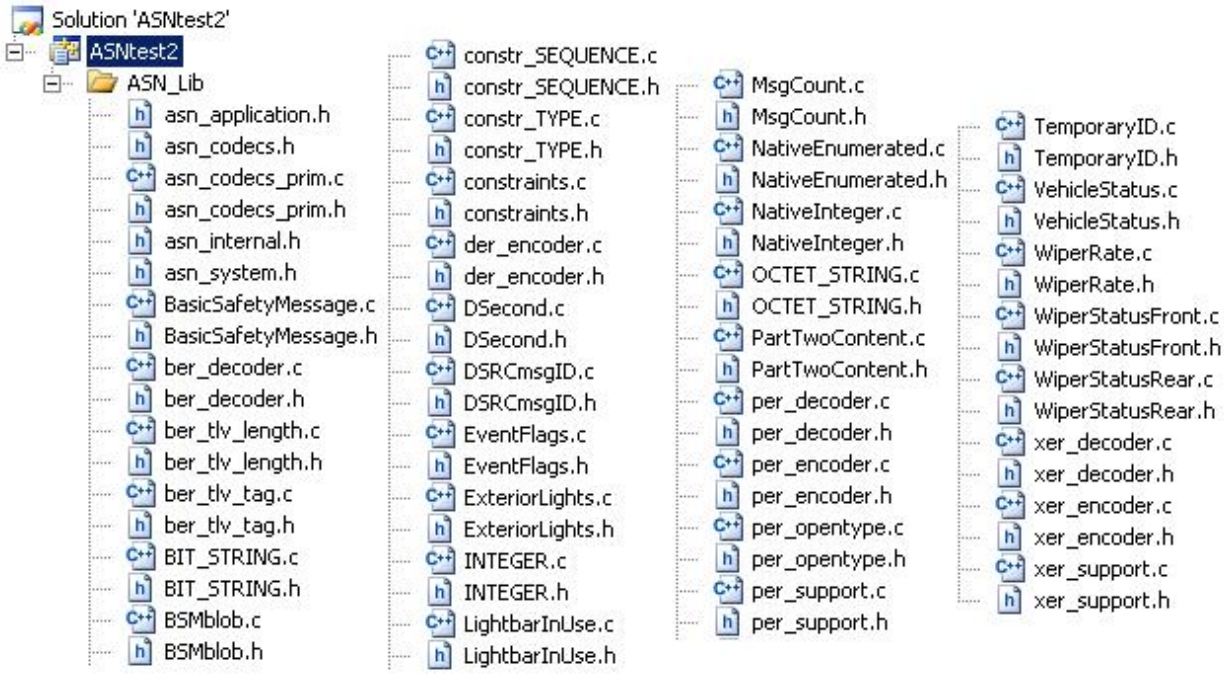

Figure 35 – ASN Tool Files Output from Simple BSM Use

The above files represent only the ASN library files produced by the ASN1c tool. While these files represent the output of the ASN1c tool, the files (and the naming used) are similar to those produced by other tool vendors. Typically these files are combined with additional files representing the application layer logic and complied to produce the final product (as can be seen in the code provided in section 8.).

In the above files are some basic support functions used in the encoding and decoding process. Files like: *ber_decode.h* or *xer_decoder.h* are examples of this. These files are full of generic functions used by the library to support the process regardless of our DSRC ASN specification. They are lightly modified each time to hold the names of all the possible PDUs[110] defined by the specification. Some, like *per_decoder.h* are not used in our DER work and could in theory be removed (but the linker step will do this for us).

[110] Every "top level" object (in our case the defined messages) is listed so that one can cast a pointer to the memory footprint to hold the encode or decode memory result.

Note the presence of some, perhaps expected, generic files (i.e., *INTEGER.h BIT_STRING.h*, *OCTET_STRING.h* etc.) and of complex types[111] like *constr_SEQUENCE.h* in the listing. Note that there are no files for the `IA5String` type listed here. This is because the BSM message does not use that type. If the complete DSRC specification was to be used, many additional files would be created including more generics.[112] Note that most of these filenames start with a capital letter, preserving the ASN rules for defined types.

The remaining other files all represent the embodiment of the ASN specification we feed into the tool. Each and every defined type (each ASN thing we created with a capital letter) has a pair of files here. And note again the initial capital letter used in the filename indicates this. For example, we have an defined type called `MsgCount` (defined as an `INTEGER (0..127)`); this is found in the files *MsgCount.c* and *MsgCount.h*. Note that instances of this object being used (such as the `msgCnt` found as the second data element in the BSM) do not produce a file. They are simply objects inside the defining class, in this case the BSM message itself which is defined in the files *BasicSafetyMessage.h* and *BasicSafetyMessage.c* . The "top level" files in this collection could be considered to be the file *BasicSafetyMessage.h* as it in turn calls all of the others, including the other defined types, which in turn link to the underlying primitive types.

Most ASN tools work in this same way. Given the source ASN specification, they produce a large number of small source files to make up the library to be used. The resulting source code library is then used by the application program (along with other source code that determines its general functionality) to build the application. The large number of created files can, at first glance, scare the user. As these files are rarely edited by the user[113] this is not really a problem, and the user becomes more comfortable with the approach once s/he learns the naming conventions being used.

6.17 The ASN.1 Encoding Tool Used in the Examples

There exist today many ASN toolsets with numerous licensing models. Some general purpose tools are sold and supported in the typical commercial software way by vendors for a fee. Other tools, typically developed by specific user communities[114] for their own needs, are also available, and many of these are open source. Even though such alternatives are low cost and the source code is freely available, the support provided cannot compare with commercial offerings. The choice of which tool to use depends on many factors besides cost, and it is beyond the scope of this guide to give such advice. Besides the business decisions of selecting the vendor and tool to be used, the primary need is that the tool encode and decode well-formed ASN messages such that the resulting *over the wire* format is interchangeable with any other compliant ASN user. The examples produced by this guide can be used as a rudimentary check of this. The reader is again directed to the resources listed in subsection 1.5 for suppliers of these tools.

In producing these guide examples, the SAE faced these same decisions. After reviewing the direction of the committee, the SAE decided to make use of the ASN1c toolkit to provide the ASN BER-DER encoding libraries needed. This is a free product (zero cost to acquire, to use, or to license for world-wide use) with no runtime restrictions, and one where the underlying source code is readily available under the GNU license for use.[115]

---

[111] In the actual ASN standard, complex types are referred to as "constructed" types, hence the naming used here.

[112] The ASN1c tool happens to ship with 106 generic files, typical for such tools.

[113] The worst thing that can be said for this process is that because each vendor differs in the details, the user eventually gets caught up in the conventions used by each tool. Loyalty to a specific tool tends to run high after each user has made the necessary investment to learn about the tool they have chosen to use.

[114] For example, one can find ASN tools specifically developed for use with Internet X.509 certificates or with the H.xxx series of telecommunications standards (H.323 H.245 H.225, etc.). Such tools typically support the specific needs of that standard better than other more general ASN options that a given standard does not use or need.

[115] See http://lionet.info/asn1c for more information and to contact its author, Lev Walkin <vlm@lionet.info>.

This toolkit has been tested and found to be suitable for the immediate needs of the guide. Because the underlying source code is available, it is possible the user community may choose to invest further in developing the tools at a later date. However, it is important to point out again that at this time, neither SAE nor FHWA has made any commitment for the further development of the code examples or tools produced as a part of this guide effort. These tools and scraps of code are provided *as is* and with no implied fitness for use.

The toolkit provides libraries to encode and decode both BER-DER and XER-XML styles of ASN and produces output for the C/C++ languages; it does not support Java at this time. It is intended for use in UNIX systems, but as part of the evaluation process, was ported to work in the MS Windows environment. At this time, some simple C applications have been built which create message content for the DSRC messages with this tool. These are covered in greater detail in section 8., and code fragments from them are used in the three application sections.

This tool has some known limitations that we have had to overcome or adapt to. It handles multiple name namespaces poorly and we needed to pre-process our ASN source code to remove names which cause duplications across the namespaces. It is known to handle large (>64k) element streams incorrectly, but these never occur in DSRC. There are problems with the tool when used with PER style encoding (another ASN encoding variant which we do not use). It seems to have issues with encoding UTC time and floating point values, which we also do not use. The constraint checking (a section of the decoder that checks that recovered data values fall within the expected ranges) seems somewhat weak. Some users feel this is important, other users do not want it at all (as it takes time and under some conditions is not needed). At this point we have not found any critical errors which would be a problem for DSRC use, but the reader should keep in mind that our application space uses only a small part of the range of ASN.

The tool is originally designed for a POSIX/UNIX operating system and some adaptations were needed to port its output into a format useable in the Windows environment (see section 8.). The tool produces simple C style files, with #include class directives if the C++ environment is found.

6.18 The ASN.1 Viewer Utility

Many of the images of decoded ASN data used in this guide have been obtained using screenshots which were taken from the COTS tool ASN1VE 1.5. This is a tool which allows decoding and building ASN messages conformant to the ASN source code supplied to it (in this case the SAE DSRC message set). This product is available from Objective Systems, Inc. (see *http://www.obj-sys.com/*) who also makes ASN toolkits and libraries similar to the one we have used. It is strongly suggested that developers equip themselves with a similar tool to view and decode ASN before attempting to debug DSRC or any other ASN message content.

7. Backward and Forward Compatibility Issues

7.1 Overview

This section deals with extensibility and how *new* or *revised* content is added to the ASN messages within the defined constructs supported by ASN. We show several ways to add new content to an existing message and recommend a few best practices to be followed. Some of the more risky ways to add such content are also discussed, although at this time none of these is used or recommended by the DSRC committee. Such new definitions is often called "local content" to reflect that a regional deployment may have to add and amend the national standard for its own needs.

The system of tagging used in the DSRC message set is called BER as mentioned in the prior section. It is a fairly simple system, with each new element in a sequence given a tag value that reflects the index position that they occupy, starting from zero. If the content of that element is simple (i.e., has no further defined inner content) then an upper byte of "8" is used, and if the content is complex (meaning that addition tagged items are defined inside it) then the upper byte is set to "A"; the net result of this is sequences of tags that appear as: `0x80, 0x81, 0x82, 0xA3, 0x84` and so on. [In this example the fourth tag (A3) is shown as complex, the others as simple.] Of course each tag is then followed by a word count (length), and then the actual byte value content of the element in the normal T-L-V style. Further details of the ASN BER tagging system can be found in section 6. of this guide.

The net result of this approach is that there is an implicit tagging in the ASN which can be easily made explicit and denoted. Consider the ASN fragment below.

```
BasicSafetyMessage ::= SEQUENCE {
   -- Header items
   msgID       DSRCmsgID,                 -- 1 byte

   -- Part I, sent as a single octet blob
   blob1       BSMblob,

   --
   -- The blob consists of the following 38 packed bytes:
   --
   -- msgCnt      MsgCount,               -x- 1 byte
   -- id          TemporaryID,            -x- 4 bytes
   -- secMark     DSecond,                -x- 2 bytes

   -- pos      PositionLocal3D,
     -- lat       Latitude,               -x- 4 bytes
     -- long      Longitude,              -x- 4 bytes
     -- elev      Elevation,              -x- 2 bytes
     -- accuracy  PositionalAccuracy,     -x- 4 bytes

   -- motion   Motion,
     -- speed     TransmissionAndSpeed,   -x- 2 bytes
     -- heading   Heading,                -x- 2 byte
     -- angle     SteeringWheelAngle      -x- 1 bytes
     -- accelSet  AccelerationSet4Way,    -x- 7 bytes

   -- control  Control,
   -- brakes      BrakeSystemStatus,      -x- 2 bytes

   -- basic    VehicleBasic,
   -- size        VehicleSize,            -x- 3 bytes

   -- Part II, sent as required
   -- Part II,
   safetyExt     VehicleSafetyExtension OPTIONAL,
   status        VehicleStatus          OPTIONAL,

   ... -- # LOCAL_CONTENT
   }
```

This example comes from the basic safety message (BSM) of the standard and is typical of the issue. We use it as the example in this section and append various additional content to it. Keep in mind that tags are already assigned by rules of ASN to this fragment. We can override the tag assignment process when we add tags manually. In such a case, the tags selected need not be the same values, although there is no reason to ever chose different tags. Here, where we seek to produce a similar result, we use the same tags. If the tagging of this message is manually added, the result is shown below (we have also removed the comments to make it shorter):

```
BasicSafetyMessage ::= SEQUENCE {
   msgID       [0] DSRCmsgID,
   blob1       [1] BSMblob,
   safetyExt   [2] VehicleSafetyExtension OPTIONAL,
   status      [3] VehicleStatus          OPTIONAL,
   ...
}
```

7.2 Issues and Choices in Adding Content

In adding any new content to an existing message the deployment faces several design decisions regarding how to add the content in such a way that devices already built (preexisting deployment population) are not harmed or are made less functional by the presence of the revised new messages in the system. If bandwidth and capacity are not stressed, then simply sending the "old" message along with any revised "new" messages may be the simple and preferred solution. This approach is always supported in DSRC (as with most message sets) but is not optimal given the limits of available channel bandwidth.

The preferable alternative is to be able to modify a message previously defined and then be able to send additional or revised content within it (the receiver is then able to skip over new tags it does not recognize and process only those that it does). The DSRC message set supports methods to accomplish this.

Within some rather limited bounds, some of the defined data elements allow for *not* sending them or for setting them to a "*not valid*" value when contained in data blobs. This also serves to support the ability to potentially revise the definitions after the fact, although the primary rational for its presence is to detect incorrect data values.

What the DSRC message set does not support is modifying an already defined message, data frame, or data element, then tagging it with the original tag and sending it (the older receiver is then clueless on how to process the value because it knows nothing of the new definition). As of the time of this writing, this is not allowed; however, as will be seen, such a process *could* be allowed if all parties agreed to some common interpretation rules regarding its use.

Taken as a whole, the methods to expand an existing message fall into one of five possible groups:

1. Adding new content with a new tag in the message
2. Adding new content after the "..." mark and let ASN worry about its tagging
3. Revising the message tag to make a new message (and hence revised content)
4. Redefining the existing content using the same tag but a new word count (not recommended for use)
5. Revise the XML definitions, adding new content to the local content stub (not covered in this document).

The first of these is the typical and recommended way to add "local" content, using tags in the reserved range of 128~255. And this also allows the standard itself to use the range 0~127 for revisions added by future editions. Note that a similar approach is used with enumerated lists as well. The next section shows examples of how this is properly done.

The second of these is the typical way the standard itself adds additional content when it is revised with any new information in a future edition. And its use is reserved for the committee's use alone. The token "..." in ASN means literally "wait; there may be more" and while it is often used with other encoding approaches (such as PER), it is also used as way to denote where extension of a message may occur in the future. Note that it typically is the last element found inside a sequence. This reflects the belief that any new content would be added *at the end*. You will also note that when this token is present, the resulting translated XML has a "local stub" element expressed in it. The ITS pattern for ASN to XML translation provides this local stub as yet another method to add local content for XML uses (which is not considered here).

The third option listed is simply to create a new message with whatever new or revised content is wanted. This method can be employed by either the local deployment (using messages numbered 128~255) or by future editions of the national standard. Of course such a new message, not existing when an older device was deployed, is ignored by that device which has no idea that it might have content that it could have an interest in. Such new messages are therefore only of value to newer devices that can be developed with an understanding of the new content. As a general design goal, it is presumed that new messages to be developed in future editions of the standard will deal with new topics and not serve as a way to revise previously issued messages in the standard.

The fourth approach is somewhat radical and is not recommended at this time. However, it is possible to accomplish and therefore is covered in sufficient detail for the reader to form his own opinion. The basic *trick* involved is to redefine an existing data element (typically an octet blob) to have a new length or word count (typically larger) and then force the receiver devices to detect this fact. When the receiver sees a data element with the tag it expects, but a word count that differs from what it expects (typically a limited range of values), then it can either ignore the content or process it to the extent possible. Observe that the length in the T-L-V has now become in fact a revision index for the item. Many users of this trick append new content "at the end" of the data blob and hence the receiver can be taught to decode up to the point the "new" content starts, and skip the rest (this rule need not be followed in all cases). This can result in an extendable set of definitions at the expense of some increased receiver complexity, but the decoding rules must be understood by all parities and cannot be changed after the fact. Because this is more or less "cheating" the ASN encoding rules, custom extensions to the ASN library used are generally needed to support it as well. Note that this process can be used on data elements as well as data frames and data blobs. More typically its use is restricted to only data blobs where there is already some precedent for encoding content which is hidden from the ASN rules.

The final choice is the preferred method of "easy" XML expansion by providing the deployment user with a schema of his own to use and edit. It is not applicable to a BER encoded system such as DSRC. Several of the other users guides (see, for example, the ATIS users guide) explain how this system is used in detail.

The next sections explain how these five methods are used in greater detail with various examples.

7.3 Adding a new data element to the message

For the purposes of example, we will add a fictitious value "myMatrix" to the message using each of the four methods suitable for ASN. Conceptually, *myMatrix* is a 9 byte value representing a 3 by 3 matrix of a motion vector contained in one byte values. Its ASN is defined as:

```
MyMatrix ::= Octet String (SIZE(9))
```

Note that from an ASN point of view there is no inner structure defined for this data element, although you the reader know a bit more about how to interpret it. Let us also presume that it has been decided that this new content needs to be sent every time the message goes out (hence part of Part I in the message). Thus, if we were defining this message from the start, we would likely place these 9 bytes into the main blob (BSMblob).

7.3.1 Case One: Local User Tags

In this case the required data element can simply be added at the end of the message, and made mandatory (not optional) so that it is sent every time. Note that the blob contents are not changed in any way. Because the added element is locally defined (and not part of a national standard) the tag value is chosen from the local range. Any value between 128 and 255 could be chosen, but by convention the numbers are assigned in ascending order.

```
BasicSafetyMessage ::= SEQUENCE {
   msgID       [0] DSRCmsgID,
   blob1       [1] BSMblob,
   safetyExt   [2] VehicleSafetyExtension OPTIONAL,
   status      [3] VehicleStatus          OPTIONAL,
   ...
   myMatrix  [128] MyMatrix,
   ...
}
```

The above change (along with having a proper definition of `MyMatrix` in the complete ASN) is all that is required. Existing decoders that follow the rule to skip tags they have not seen before (that were not defined when they were implemented) can read and use this message without fault or harm. Newer devices, those that know about the content that tag 128 represents, can decode the additional content was well.

> Keep in mind that most ASN BER decoder designs can read, and recover, any well-formed BER encodings; they do not need the ASN specification to which that encoding relates. Having this information provides additional decoding details that can be used in the process, but the fundamental ability to detect, process, or skip a T-L-V entry is usually always present.

7.3.2 Case Two: Revised National Message Set, new data element

In this case the required data element can simply be added at the end of the message, and made mandatory (not optional) so that it is sent every time. Note that the blob contents are not changed in any way. Because the added element is part of the (new and revised) national standard, the tag value is chosen from the next available number or the ASN can assign it directly (implicit encoding);. the result is the same. Note the use of a second "..." because this edition may itself have new content added to it at some future date.

```
BasicSafetyMessage ::= SEQUENCE {
   msgID       [0] DSRCmsgID,
   blob1       [1] BSMblob,
   safetyExt   [2] VehicleSafetyExtension OPTIONAL,
   status      [3] VehicleStatus          OPTIONAL,
   ...
   myMatrix    [4] MyMatrix,
   ...
}
```

or alternatively:

```
BasicSafetyMessage ::= SEQUENCE {
   msgID       DSRCmsgID,
   blob1       BSMblob,
   safetyExt   VehicleSafetyExtension OPTIONAL,
   status      VehicleStatus          OPTIONAL,
   ...
   myMatrix    MyMatrix,
   ...
}
```

The above change (along with having a proper definition of `MyMatrix` in the complete ASN) is all that is required. Existing decoders that follow the rule to skip tags they have not seen before (that were not defined when they were implemented) can read and use this message without fault or harm. Newer devices, those that know about the content that tag represents, can decode the additional content was well. Note that from the receiver's point of view, there really is no difference between this case and the prior one.

This method is reserved for use by the committee itself as it adopts new editions of the standard.

7.3.3 Case Three: Revised National Message Set, new message

In this case, the required data element can simply be added at the end of the message, and made mandatory (not optional) so that it is sent every time, AND the message itself is given a new name and assigned a new numbering. Thus the old message and the new coexist as valid messages in the set. Note that the blob contents are not changed in any way. Because the added element is part of the (new) national standard, the tag value is chosen from the next available number or the ASN can assign it directly (implicit encoding); the result is the same. Note that use of a second "..." because this edition may itself have new content added to it at some future date. The critical change here is the name of the message to contain "2", but other modifications to the message structure could also be made at this time (the new message need not follow the old message pattern).

```
BasicSafetyMessage2 ::= SEQUENCE {  --   <- Note the "2" here
   msgID       [0] DSRCmsgID,
   blob1       [1] BSMblob,
   safetyExt   [2] VehicleSafetyExtension OPTIONAL,
   status      [3] VehicleStatus          OPTIONAL,
   ...
   myMatrix    [4] MyMatrix,
   ...
}
```

or alternatively

```
BasicSafetyMessage2 ::= SEQUENCE {
   msgID        DSRCmsgID,
   blob1        BSMblob,
   safetyExt    VehicleSafetyExtension OPTIONAL,
   status       VehicleStatus          OPTIONAL,
   ...
   myMatrix     MyMatrix,
   ...
}
```

In addition, the message ID numbering of the messages must also be amended to add this message as shown below:

```
DSRCmsgID ::= ENUMERATED {
   reserved                      (0),
   alaCarteMessage               (1),  -- ACM
   basicSafetyMessage            (2),  -- BSM, heartbeat msg

   basicSafetyMessage2           (17),  -- New Message here

   basicSafetyMessageVerbose     (3),  -- used for testing only
   commonSafetyRequest           (4),  -- CSR
   emergencyVehicleAlert         (5),  -- EVA
   intersectionCollisionAlert    (6),  -- ICA
   mapData                       (7),  -- MAP, GID, intersections
   nemaCorrections               (8),  -- NEMA
   probeDataManagement           (9),  -- PDM
   probeVehicleData              (10), -- PVD
   roadSideAlert                 (11), -- RSA
```

```
    rtcmCorrections                 (12), -- RTCM
    signalPhaseAndTimingMessage     (13), -- SPAT
    signalRequestMessage            (14), -- SRM
    signalStatusMessage             (15), -- SSM
    travelerInformation             (16), -- TIM

    ... -- # LOCAL_CONTENT
    }
    -- values to 127 reserved for std use
    -- values 128 to 255 reserved for local use
```

Or if a locally defined message type, it would alternatively be defined as:

```
    basicSafetyMessage2             (128),  -- New Local Message here
```

The above changes (along with having a proper definition of `MyMatrix` in the complete ASN) is all that is required. Existing decoders will not be aware of this new message type and be unable to decode it (although they can still parse over the T-L-V elements). Newer devices, those that know about the message, can decode the content. Note that from the receivers point of view there really is no difference between the local and national alternatives shown here.

7.3.4 Case Four: Indexed Word Count method

This is by far the most complex alterative of the lot. In this case we modify the contents of the BSMblob to add these 9 bytes "at the end" of the existing octet definition. The new blob definition is shown in the typical ASN comment form, and a new ASN entry would also be created and placed into the document. Observe that this trick can be used on any element and that the static nature of the existing content (rather then redefining it) is simply a convenience to provide a simpler example; all these additional modifications could be done.

As a result of this change, all receiver devices must be able to detect and cope that the recovered word count is no longer always 38 bytes, but is now either 38 or 47 bytes and react in one of two ways defined ways:

a) Process the first 38 bytes presuming that the content remains as previously defined, then discard the remaining additional bytes (in this case 9 of them)

b) Process all 47 bytes, presuming that the initial content remains as previously defined, and that the next 9 bytes represents the new (`myMatrix`) content to be extracted. Discard any additional byte content found (which would represent yet a newer revision).

Older receivers would take the first path, while newer receivers would use the second. The resulting ASN is shown below:

```
BasicSafetyMessage ::= SEQUENCE {
   -- Header items
   msgID       DSRCmsgID,                  -- 1 byte

   -- Part I, sent as a single octet blob
   blob1       BSMblob,

   --
   -- The blob consists of the following 49 or 38 packed bytes:
   --
   -- msgCnt      MsgCount,               -x- 1 byte
   -- id          TemporaryID,            -x- 4 bytes
   -- secMark     DSecond,                -x- 2 bytes

   -- pos      PositionLocal3D,
     -- lat       Latitude,               -x- 4 bytes
     -- long      Longitude,              -x- 4 bytes
     -- elev      Elevation,              -x- 2 bytes
     -- accuracy  PositionalAccuracy,     -x- 4 bytes
```

```
-- motion   Motion,
  -- speed     TransmissionAndSpeed, -x- 2 bytes
  -- heading   Heading,              -x- 2 byte
  -- angle     SteeringWheelAngle    -x- 1 bytes
  -- accelSet  AccelerationSet4Way,  -x- 7 bytes

-- control  Control,
-- brakes      BrakeSystemStatus,    -x- 2 bytes

-- basic    VehicleBasic,
-- size        VehicleSize,          -x- 3 bytes

   -- Present in Revision XX onwards
   -- myMatrix   MyMatrix              -x- 9 bytes

-- Part II, sent as required
-- Part II,
safetyExt     VehicleSafetyExtension OPTIONAL,
status        VehicleStatus          OPTIONAL,

... -- # LOCAL_CONTENT
}
```

The above change (along with having a proper definition of `MyMatrix` in the complete ASN and revising the actual blob definition to include the new item) is all that is required. In addition, we will need to come up with some acceptable way to denote that this item can now have either 38 or 47 bytes as the only allowed "valid" numbers. Again, observe that the length count of the T-L-V has become a sort of *index* into how to decode the value. (This is a tech-pubs issue we can decide later should we ever go this route.) Unaddressed is what action to taken when *myMatrix* is not longer being sent and some other new element is now to be added. It may be that this approach only allows concatenation of new content and not its subsequent deletion.

The biggest change is a presumption of a new complex decoding rule on decoders, which would need to be placed on ALL decoders being designed today, so that in the future this rule could be invoked without causing them to fail on such messages. It is beyond the scope of this paper to estimate how big an impact this would in fact be in software development, but is presumed to be large. The new development scale could be sharply reduced by allowing this "trick" to be invoked ONLY on octet blobs.

Again this method is not recommended for use at this time.

7.4 Concluding Remarks

The above discussion has illustrated the four known ways to extend the message set defined in the DSRC standard. Of these, the first three are well known and supported by the standard as it now stands. The fourth method is considered experimental and is not recommend for use as there are no receivers currently being developed to support the presumed additional decoding logic rules which it requires. The fifth method is used only for XML and has no analogous equivalent in the BER encoding used in DSRC. Users who feel a need to apply these methods are urged to contact the DSRC committee to the allow coordination with the current revise efforts.

8. Tools for Encoding Examples

8.1 Basic Use of the Tools

The DSRC Guide tool performs three basic missions as follows:

1. To view, create, and edit messages constructed using the rules and requirements outlined in the SAE J2735 data dictionary message set.
2. To allow the user to explore and see the actual code that implements the DSRC functionality as a learning aid (and using the chosen ASN1c tool library).
3. To form an open source “reference tool” that can be used by all to read and validate messages created by any tool; in short, a common truth source to compare messages for interoperability needs.

In this section of the guide we discuss how the software used to create the tool operates.

This topic may not be of interest to all readers and can be safety skipped by those with no desire to ever examine the source code itself. Those interested in coding their own work or in reusing the ASN library will find this section informative as supplemental documentation to the source code.

8.2 Internal Structure of the Utility Application

There are three major parts to the code base used to create the tool

The ASN Encoding Library Which comes from the output of the ASN.1 tool we are using and provides the many small files that make up the handlers for each message type and its elements. You will need such a tool and may select another vendor, but will undoubtedly end up with many similar small files.

The Message Processing Code Which does the actual assembly and encode/decode of the message data. These routines are what we deal with in most of the guide code fragments, and they in turn call the routines above to do the serialization work.

The User Interface — A minimal implementation of a Windows application used to display and process the data and call the message processing routines. Because MS Windows itself has no place in a real-time system, the technical aspects of this part of the program are not emphasized in the guide.

Developers who want to create their own product will want to borrow from the first two sections (essentially ignoring the third). If another ASN compiler tool is used, then it will likely provide its own ASN encoding library for use. Advanced users may wish to study the ASN encoding library source code itself (which is also freely available under the GNU license) to devise libraries which meet their own unique needs.

Each of these three areas are expanded further in the following subsections.

8.2.1 ASN Encoding Library

The ASN encoding library consists of over 560 short source code files which collectively make up and implement the library which was produced from the original ASN source code of the SAE J2735 standard. Each and every type definition is implemented as its own C++ and h file. The naming of these files follows the names given by the type definitions in the ASN source file.

For example, the first of these [the files *Acceleration.cpp* and *Acceleration.h*] implements the type "Acceleration" from clause 7.1 of the standard, which has the ASN of:

```
Acceleration ::= INTEGER (-2000..2001)
   -- LSB units are 0.01 m/s^2
   -- the value 2000 shall be used for values greater than 2000
   -- the value -2000 shall be used for values less than -2000
   -- a value of 2001 shall be used for Unavailable
```

This is of course just a simple integer, and it in turn uses the routines found in a "static" file *NativeInteger.h* which in turn imports *Integer.h* to together implement its behaviors. These static files (about 50 in all) are combined with the files following your chosen type naming to form the library. There are files for "each" basic type of ASN structure as well as some supporting "`codecs`" that implement the different encodings.

Each ASN type has "`_t`" added to the ASN name to become a C language variable type, in this case: `Acceleration_t`. The resulting descriptor definition (the function used to call operations on this type) has "`asn_DEF_`" front of it, in this case: `asn_DEF_Acceleration`. Thus, to create an instance or a pointer to this type, you create lines like:

```
Acceleration_t   MyCopy;      // An instance of the type
Acceleration_t*  MyCopyPtr;   // A pointer to the type
```

In the *Acceleration.cpp* file (and in every similar file) you will find the code needed to implement the basic ASN functions described on the next page. Also in this file is the "constraint checking" code used to determine if the value contents (size, length, values, etc.) of the instance are within allowed bounds. In this particular implementation of ASN the constraint checking is separate from the decoding operations. This is generally considered a good thing because it allows the programmer to determine and control when processing effort is expended to evaluate constraints.[116] In the case of the Acceleration example, the range checking starts on line 26 and is shown below. Note that

[116] It is foreseen that constraint checking will not be necessary or likely on every received message. In a fashion similar to security certificate validation, this becomes necessary only when the trust level for a received message is low (such as the first time a new user is encountered). Given a source of messages that has proved valid in the past, a deployment may forego this checking when sufficient trust has built up. This is not to say that an ill-formed message would not be detected and rejected; such a message will not be able to be decoded, a necessary step before constraint checking can take place. See the file *MSG.cpp* for some examples of this. The lower layers may also detect some message transmission errors.

the tool provides the source code file name and line number where the constraint error was detected, which is fine for our needs but which may required additional logic for a production application. The tool also allows the programmer to provide a callback routine to implement such functionality.

```
if((value >= -2000 && value <= 2001)) {
    /* Constraint check succeeded */
    return 0;
} else {
    _ASN_CTFAIL(app_key, td, sptr,
        "%s: constraint failed (%s:%d)",
        td->name, __FILE__, __LINE__);
    return -1;
}
```

More complex types implement this same sort of basic logic in sequences, once for every element found in them. They, too, then call the "static" routines to implement each actual element from its basic type. In this way complex structures are built up from simple parts.

The tool provides seven basic calls for every type that is defined by the source ASN. Recall that a "type" is any definition that has been created, not just the messages, so these calls can be used for almost every element found in the standard. The ASN1c documentation provides the following description about these calls, which is repeated here:

ber_decoder
This is the generic restartable BER decoder (Basic Encoding Rules). This decoder will create and/or fill the target structure for you. Please refer to section *Decoding-BER* in the ASN1c documentation.

der_encoder
This is the generic DER encoder (Distinguished Encoding Rules). This encoder will take the target structure and encode it into a series of bytes. Please refer to section *Encoding DER* in the ASN1c documentation.

xer_encoder
This is the XER encoder (XML Encoding Rules). This encoder will take the target structure and represent it as an XML (text) document using either BASIC-XER or CANONICAL-XER encoding rules. Please refer to section *Encoding XER* in the ASN1c documentation.

xer_decoder
This is the generic XER decoder. It takes both BASIC-XER or CANONICAL-XER encodings and deserializes the data into a local, machine-dependent representation. Please refer to section *Decoding XER* in the ASN1c documentation.

check_constraints
Checks that the contents of the target structure are semantically valid and constrained to appropriate implicit or explicit subtype constraints. Please refer to section *Validating the target* in the ASN1c documentation.

print_struct
This function converts the contents of the passed target structure into human readable form. This form is not formal and cannot be converted back into the structure, but it may turn out to be useful for debugging or quick printing. Please refer to section *Printing the target* in the ASN1c documentation.

free_struct
This is a generic disposal which frees the target structure. Please refer to section *Freeing the target* in the ASN1c documentation.

Each of the above functions takes the type descriptor (asn_DEF_...) and the target structure (a pointer to an instance of the type definition) as inputs.

In this project we use the following calls:

ber_decoder	Used in the "ok" routines of all dialogs
der_encoder	Used in the dialog *MSG.cpp* for incoming messages
xer_encoder	Used in the "ok" routines of each message dialog
check_constraints	Used in the dialog *MSG.cpp* for incoming messages
print_struct	Used in the "ok" routines of each message dialog
free_struct	Used in various places to de-allocate memory

These are each called by passing the type (the `asn_DEF_name` value) and a pointer to the structure in question. Some of the routines also need an array of bytes to read into or out of. The routines can be called directly from their type definitions, or by their global counterparts (we typically do the latter in the example code). For example, creating the Basic Safety Message (BSM) in BER from a structure in memory is called as:

```
result = der_encode_to_buffer(
          &asn_DEF_BasicSafetyMessage,  // the ASN type
          theMsgStruc,     // a ptr to the struc to read
          theBuffer,       // the byte array to build
          MAX_MSG_SIZE);
```

While decoding the array of bytes back into the structure use this call:

```
result = ber_decode( 0,    // a restart value (not used)
          &asn_DEF_BasicSafetyMessage, // the ASN type
          theMsgStruc,     // a ptr to the struc to build
          theBuffer,       // the byte array to read
          theBufferSize);
```

When the time comes to free the BSM structure in memory, a deallocate routine is called using a macro as:

```
ASN_STRUCT_FREE(
          asn_DEF_BasicSafetyMessage, // the ASN type
          theMsgStruc);    // a ptr to the struc to free
```

In general, the ASN library has been found to be acceptable, bug free, and robust without the need for further changes when used for DSRC. One exception to this, due to the needs of the Microsoft C compiler, is the use of nested structural definitions. You will see that in many of the ".h" files, all nested structures have been defined with each new structure independently. This can be easily found by seeking for the comment `//DCK mods` in the source code. Another change required by the Microsoft C complier is the treatment of enumerated values. These have all been edited to make every enumeration set member a unique and global value in order to meet the variable scope needs of the .NET environment.

If you wish to start fresh with a working ASN library where these changes have all been made, the contents of the folder *\ASN_lib_r36* contains the complete set of all ASN files you would need to use. Alternatively, you can run any ASN source specification of your own choosing through the ASN1c tool to create your own libraries.

8.2.2 Message Processing Code

The message processing code handles the user interface of the dialogs and then assembles the final structure from the user's inputs and calls the *encoder* or *decoder* as described above. The bulk of this code is concerned with updating and maintaining the user displays and visual display of the code contents on the screen. Needless to say, in an embedded system much of this is not needed or wanted and the code can be reduced to just the structural assembly calls needed to create the ASN contents. There is a number of basic code patterns repetitively used in this section and these are now considered to provide a further overview of how to understand the source code.

The general flow of the code is to have one dialog window for each data frame or message. In this dialog, each individual data concept is then represented by one control (if the data element is simple or atomic) or by another nested dialog (for data frames and complex content). In this way a series of nested dialogs is used to build up the complete message and allow data entry to each atomic data element in turn. When each dialog is invoked, data is passed into it from the caller, and returned with whatever changes are made when done. The data exchanges follow the exact definitions of the ASN memory structure, which in turn comes from the ASN source documents defined by the SAE DSRC standard.

With the exception of the messages themselves (which are not modal) all such dialogs are shown in a "modal" form which requires the user to enter data on that dialog before proceeding to another location or dialog. As data is built up in each visual display, the controls show the resulting encoding bytes for that element and the data frame that contains it to the right of each control. This interactive effect uses common Windows messaging to cause the visual controls to be updated as data is entered or changed.

While the tool today allows creating only three representative message types, the format and styles used can be readily extended to other messages to complete the rest of the message set. An exception to this is the abilities of the "read messages" dialog which can today read and validate all 16 of the existing DSRC messages defined in the current standard.

The basic flow of every dialog's lifetime is as follows.

#1 Create an instance of the dialog

#2 Import data into the dialog (passing data into the dialog)

a. Copy a pointer to the structure or data frame used in the dialog OR
b. Copy a blob of bytes representing the data used in the dialog

#3 Initialize the dialog, taking the passed in data and assigning it to various controls

a. For each control in the dialog
    i. Install the passed in data into the control, if any is present
    ii. Create a temp "empty" copy if one does not yet exist as a working place holder for any new user data we may obtain
    iii. Cause the visual aspects of the control to be updated

#4 Run the dialog, responding to user inputs in each of its controls

a. When a control is activated with new data or a mouse press
    i. Focus the window on that control (done by windows)
    ii. Update the control value and the visual representation of that control from the user input
        1. Inform the user if data is not valid or allowed
        2. invoke the BER encoding on this item, if needed
    iii. Update the data frame in which the control is used
        1. invoke the BER encoding on this item, if needed
    iv. Update any other control values that may be affected
        1. invoke the BER encoding on this item, if needed

#5 When the dialog is dismissed, return control to the caller

a. If the user pressed "ok" use the new data from the dialog
b. If the user pressed "cancel" discard the new data from the dialog

#6 Export data from the dialog back to caller (returning data from the dialog)

a. The reverse of step #2 above

#7 Dismiss (destroy) the dialog and clean up any unneeded heap items

The above steps are implemented in a Windows environment, which from a practical point of view means that each step tends to be a separate function call and these functions are invoked as events occur (mouse movements, keyboard input, etc.). Because the dialog is built (derived) from the CDialog class, most of the functionality is hidden in the underlying class and we need only be concerned about our controls. Event messages to these controls are generated by the Microsoft framework which also handles routing the resulting messages to the correct control. Each of these steps will be briefly examined in a typical dialog.

In addition to the above routines, the basic message dialogs also have routines to serialize the final ASN into a BER or XML encoding. These routines are basically the same as the code used to create the overall data frame display in each dialog, but have some additional functionality in order to allow saving the message and to express it in an XML format as well.

A final few words are spent examining the process of reading in a raw BER message and calling constraint checking on the resulting structure.

The above steps are used for *all* the dialogs with only minor variants as needed. Once you have mastered understanding one dialog, the others will appear very similar.

For the following code examples we use code from the "full position vector" [file: *fullPositionVect.cpp*] and from the RoadSideAlert message that calls it [file: *RSA.cpp*] as well as the blob-style dialog for the BSM "blob1" data frame [file: *BLOB1.cpp*]. Again, the examples and patterns shown are typical for all the code in the project.

Step #1 Creating an instance of the dialog class and all its routines is done simply by creating a C language variable of the dialog in question. A few examples are shown below. Note that the actual name used here for the dialog itself is chosen by the programmer, and is not provided by the ASN library (as the dialog is created entirely by the user).

```
BLOB1           theBLOB1Dlg;      // create an instance of the dialog
FullPosVectDlg  theFullPosDlg;    // create an instance of the dialog
WiperStat       theDlg;           // create an instance of the dialog
```

This code is typically found in a button press routine which opens the dialog and then processes the results when completed. The class "create" and "destroy" functions are called at this time as well, however these are typically null functions. Some additional opening and window placement logic is used in the three basic windows (not shown here, see file *wordpad.cpp*[117] around line 1297 for an example in the function `OnBSM1Start()`). Most dialogs are simply opened directly on top of the underlying window which owns them. When the dialog returns, the return code is checked and the dialog's data is either used or discarded. The overall framework of this logic is shown below for the full position vector (FPV) dialog which is being opened from within the Roadside Alert code.[118] Note also the frequent use of a utility function called `AddRtxtReport` to append information to the log file and inform the user precisely where we are and what is occurring in the logic of the program.

```
void RSA::OnBnClickedButton_FPVector()
       {
       FullPosVectDlg  theFullPosDlg;
       // create an instance of the dialog (step #1)
       // inform user what is occuring
       theApp.log.AddRtxtReport (enInformative,
              "The Full Position Vector Dialog has been invoked from the RSA dialog.",
              true);
```

[117] The file *wordpad.cpp* is the source code file most like a traditional "main" file in this program. It is mostly concerned with the Notepad source code example but also deals with invoking the menu handling process that fires off each message dialog. The function *OnBSMStart* is used to start the Basic Message dialog up, transferring control to routines in the file *BSM.cpp* for processing. Similar files with similar names are used for the other messages.

[118] The actual code found in the source contains some additional (commented out) lines not repeated here which are useful for debugging the passed structures.

```
    theFullPosDlg.m_Pos = m_pPos;
    // assign current data into the dialog bt ptr (step #2)

    // invoke dialog, (step #3)
    // processing it further if we get an ok return
    // (step #4 occurs inside the dialog code, then step #5 returns)
    if ( theFullPosDlg.DoModal( ) == IDOK ) {
        // get the new data back from the dialog
        // note that any validity testing logic
        // was done inside the dialog box
        m_pPos = theFullPosDlg.m_Pos;
        // Assign result back to the caller (step #6)

        // Now use new bytes now, updating this dialog display
        // We update the displayed data we got back.

        // inform user what happened
            theApp.log.AddRtxtReport (enInformative,
                    "The Full Position Vector Data has been updated." true);
    }
    else {
        // inform user what happened
        theApp.log.AddRtxtReport (enInformative,
                  "The Full Position Vector Dialog has been canceled (no change to
                  data).", true);
    }
} // At this point the dialog goes out of scope and step #7 occurs
```

Step #2 Before the modal dialog is created and displayed (but after it has been declared) we must send any relevant data into it. This is done by either a bulk copy of the bytes involved (in the case of octet blobs) or by passing a pointer to the defined memory structure to be used (which is more typically used).

Exchanging a pointer is rather simple, one simply passes the address to the dialog's member variable as was shown in the above code.

```
theFullPosDlg.m_Pos = m_pPos;     // the FPV case
theDlg.m_pWipers = m_pWipers;     // the wipers case
```

Bulk copies of bytes are a bit harder because a loop is used for each direction (in a routine called `Stuffit`), and afterward a routine called `UnPack` or `Pack` is called to exchange the byte fields with assign variables in the dialog class. The BLOB class is the best example of this mode.

```
theBLOB1Dlg.StuffIt(m_blob1);     // by the caller
```

Which is in turn implemented with the below. The flag "need unpack" triggers the step #4 process to occur.

```
// move individual data elements values to the blob
void BLOB1::StuffIt(BYTE* pTheInBytes)
{
    // copy bytes to from passed pointer to local vars
    int i;
    for (i=0; i < BLOB1_SIZE; i++) {
        // the new data is copied into the dialog member
        m_blob1[i] = pTheInBytes[i];
    }
    m_needUnPack = true;
    // set the trigger flag so that an UnPack() will be called
}
```

The unpack routine in turn takes data from the blob array and installs it into the various variables. Here is a fragment dealing with the latitude data element:

```
// do the Latitude (lat) data element
offset = offset + 2;
pRawLongData = (long *) &m_blob1[offset];
          // treat first 4 bytes in the array as a long
theApp.util.Swap4bytes((BYTE *)pRawLongData);
          // reverse bit order here
tempValue = *pRawLongData;
m_latitude = tempValue;


_itoa_s ( m_latitude, tempString, 100, base);
   // convert to a base X string
pControl = (CEdit*) GetDlgItem (IDC_EDIT_lat);
pControl->SetWindowText(tempString);  // write value to control
```

Step #3 involves initializing the dialog and takes place (calling the `DoModal` function for the class) when the first function of the class is called, in this case as an effect of the `theFullPosDlg.DoModal` call in the above code. When this line is reached, the `OnInitDialog` function for the class is called, after which control passes to the basic dialog class routines that manage the flow of control and events during the dialog's life. (These are part of the dialog's base class and are not normally modified by the programmer. Most programmers can in fact ignore this completely once an understanding of the flow of events is gained.)

The init routine first builds various controls which are needed by the dialog (creating combo box choices and other similar steps); it then processes the data needed by each control in that dialog. Here is an example fragment for the optional "elevation" data element found in the full position vector. Note that in this example some byte manipulation is taking place to handle the *big-endian* nature of ASN encoding. This example is using a declared data element rather than a pointer. Pointers are typically used when optional content is defined in the ASN (and when the pointer is set to NULL the content is not present).

```
// if item is present (if pointer is not null),
// copy it over and set checkbox
CButton* theCkBox;
theCkBox = (CButton*) GetDlgItem (IDC_CHECK_elev);
if (m_Pos->elevation != NULL)
{
   theCkBox->SetCheck(BST_CHECKED);
   m_elevation = ((m_Pos->elevation->buf[0])<<8 ) +
                              (m_Pos->elevation->buf[1]);
   // write to the edit label
   pCont = (CEdit*) GetDlgItem (IDC_EDIT_elev);
   text.Format("%i", m_elevation);
   pCont->SetWindowText(text);
}
else
{
   theCkBox->SetCheck(BST_UNCHECKED);
   m_elevation = 0; // a value not a pointer
}
```

Perhaps a more typical example is when the heap space needed for an optional element must be allocated, as the below code for the optional rear wiper sweep rate (in the wipers data frame) shows.

```
pCheck = (CButton *) GetDlgItem (IDC_CHECK_rearRate);
if (m_pWipers->rateRear != NULL)
{  //use input to set value here;
   text.Format("%i", *(m_pWipers->rateRear));
   pText = (CStatic*) GetDlgItem (IDC_EDIT_rearWiperRate);
   pText->SetWindowTextA ( text );
   pCheck->SetCheck(BST_CHECKED);
}
else
{  // create an empty struc we can use here
   m_pWipers->rateRear = new WiperRate_t;
   *m_pWipers->rateRear = 0;
   pCheck->SetCheck(BST_UNCHECKED);
}
```

Note that this code also is making use of the behavior of the Microsoft framework. It in fact never needs to set the class variable `m_pWipers->rateRear` when the data is found to be present (the true side of the if statement above). It depends on the framework calling the routine `OnBnClickedCheck` as a side effect of changing the user control checkbox `IDC_CHECK_rearRate`. This results in the following function being run, which then sets the variable (by calling `OnEnChangeEdit_rearwiperrate`) and updates the display labels as needed (this crosses into step #4 to some degree).

```
void WiperStat::OnBnClickedCheck_rearrate()
{
   // if checked...
   CButton* pCheck = (CButton *) GetDlgItem (IDC_CHECK_rearRate);
   if (pCheck->GetCheck() == BST_CHECKED)
   {  // make heap entry if needed
      if (m_pWipers->rateRear == NULL)
      {  // create a struc we can use here
         m_pWipers->rateRear = new WiperRate_t;
         *m_pWipers->rateRear = 0;
      }
      // if text is set to nothing, make it a zero now
      CString userInput;
      CEdit* pControl = (CEdit*) GetDlgItem
                                        (IDC_EDIT_rearWiperRate);
      pControl->GetWindowText(userInput);
      if (userInput == "") pControl->SetWindowText("0");
      // update the entry now using current text
      OnEnChangeEdit_rearwiperrate();
   }
   else
   {  // if not checked, delete off heap.
      if (m_pWipers->rateRear != NULL)
               delete m_pWipers->rateRear;
      m_pWipers->rateRear = NULL;
      PaintComposite();
   }
}
```

The routine `OnEnChangeEdit_rearwiperrate` is then called to deal with the current data (and is discussed in step #4 below).

Note that taken as a group, this creates and disposes of the heap allocation for the optional element (rear sweep rate in this case) as the user changes the value in the control and enables or disables the checkbox to indicate if it should be present or not. Every optional data concept value has this same heap management issue, but the precise placement of heap allocation and deallocation can vary based on programmer preferences for such things. Items which are not optional are created as a side effect of declaring the underlying object which contains them.

This process of "installing" the variables is repeated for each control to establish its initial value settings; the dialog is then "opened" and presented to the user.

Step #4 Once the dialog is open and being displayed, user events then drive the focus to each control and allow the user to enter data for it. Each time the data changes, an "on change" function for that control is called and the new information is processed. Note that this is not particularly efficient from a processing point of view because a keyboard entry for a value like "1000" results in the same code being processed four times.[119] The overall flow looks like the below code (in this case taken from the file *WiperStat.cpp*).

```
void WiperStat::OnEnChangeEdit_rearwiperrate()
{
    #define rateMaxValue 127 // units of sweeps per min
    #define rateMinValue 0
    CString userInput;      // the users typed input string
    long userValue;         // user input as a number
    CStatic* pText;         // ptr to a labels text
    CString userInputHU;    // human readable format
    CString userInputRV;    // actual hex value format
    CEdit* pControl = (CEdit*) GetDlgItem (IDC_EDIT_rearWiperRate);
    pControl->GetWindowText(userInput);
    userValue = atol(userInput);
    if ((userValue <= rateMaxValue) &&
        (userValue >= rateMinValue)) {
        // is valid input (in range), process it
        userInputHU.Format ("%i s/min", userValue );
        userInputRV.Format ("0x%2.2X", (BYTE)(userValue));
        pText = (CStatic*) GetDlgItem(IDC_STATIC_rearWiperRate_HU);
        pText->SetWindowTextA ( userInputHU );
        pText = (CStatic*) GetDlgItem(IDC_STATIC_rearWiperRate_RV);
        pText->SetWindowTextA ( userInputRV );
        if (userValue !=0)  // be sure checkbox is on
        {
            CButton* theCkBox = (CButton*) GetDlgItem
                        (IDC_CHECK_rearRate);
            theCkBox->SetCheck (BST_CHECKED);
        }
    }
    else {
        // warn user, is out of range
        CString text;
        text.Format( "Value entered [%s] is out of range for this data element. \n\nAn
integer between the limits of %i and %i is needed to be valid. ", userInput,
rateMinValue, rateMaxValue );
        MessageBox (text, "Inalid Data", MB_OK | MB_ICONWARNING);
    }
    // set the data element for this item in this class
    *(m_pWipers->rateRear) = (BYTE)userValue;
    PaintComposite();  // cause data frame to be updated
}
```

At the bottom of this routine is a function called `PaintComposite` which handles step 4.a.iii, updating the data frame in which this data element lives (generically called a blob in the code). This routine (or a similar one, the precise name in other dialogs varies) is called by all the controls to update the data frame when they change their contents. It simply calls the BER serialize function on this subpart of a message and then displays the results at the bottom of the dialog (into a label called `IDC_STATIC_blob_RV`) with code like that shown below.

[119] Once for the value "1" then once again for "10" then again for "100" and finally for "1000". If the value "1000" is pasted into the control (rather than entered one keystroke at a time), then the processing is only performed once.

The wiper case is a more complex example of this because ”wiper” does not exist as a defined ASN type directly (wipers is a part of the `VehicleStatus` object). So in this case we must construct a larger ASN structure which contains the wiper portion and then extract the wipers content. This is not typical; most of the time the type exists directly (a side effect of design decisions made in the data dictionary). Here is a typical example taken from the file *fullPositionVect.cpp* to update the FPV data frame display in its dialog.[120] Note that it calls the serialize routine in the normal way.

```
void FullPosVectDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    //update the output text at IDC_STATIC_blob_RV
    // based on current member vars m_pTemp points to the FPV struc
    CString text;
    CStatic* pText = (CStatic*) GetDlgItem (IDC_STATIC_blob_RV);
    if (m_pTemp == 0)
        pText->SetWindowTextA ( "No data present." );
            // Should never happen, as some data is in fact required
    else
    {
        // Rebuild the temp struc here
        FillOutFPV(m_pTemp);
        // Convert the FPV struc to its encoded value
        BYTE theArray[100];
        asn_enc_rval_t er; // Encoder return value
        er = der_encode_to_buffer(&asn_DEF_FullPositionVector,
            m_pTemp, theArray, 100);
                    // serialize into BER bytes the struc
        if (er.encoded >= 100)
                    MessageBox("Increase buffer in FPV encode.!",
                            "Buffer overflow!",MB_OK+ MB_ICONWARNING);
        if(er.encoded == -1) {
            // Failed to encode the data, report error
            text.Format("Cannot encode %s: %s\n",
                            er.failed_type->name, strerror(errno));
            theApp.log.AddRtxtReport (enError, text, true);
            pText->SetWindowTextA ( text );
            }
        else {
            // Used returned number of bytes for byte string
            pText->SetWindowTextA (
                            theApp.util.FormHexString(theArray,
                                    (int)er.encoded,3,false,12) );
            }
    }
}
```

In a few cases we also need to update the value in another field due to the change. The CRC data element used in the Roadside Alert (RSA) message is an example of this.[121] Here we use a timer value to cause this data element to be updated. When the timer times out (100 mSec after the last input is processed), we then update the data element, giving the visual appearance of an instant update. The key code that triggers this in each control update routine is shown below and simply sets a timer countdown value.

```
OnStartCRCTimer(); // update CRC
```

Then, when the timer elapses (creating another event message) we run the routine `OnBnClickedButtonDoCRC` which is similar to all the other “on change” routines and simply updates that control.

[120] The term “paint” is a common term used in windows programming to update the screen with data. Many windows programs have update routines that refers to “painting the screen” this way.

[121] As the value is based on an algorithm calculated over the other bits in the final message.

At this point the data and the structure used to contain that data represents what this dialog contains. If the user were to quit the dialog with an "ok" button return, there is no further work to be done, we simply can return this structure (by means of a pointer or by bulk copies) to the caller. The may be some unused memory to clean up, but that is about all there is to it.

In **step #5** the user has dismissed the dialog box. This can occur one of two ways, the "ok" button may have been pressed, or the "cancel" button may have been pressed.[122] The resulting code returned to the caller can be used to determine this. Referring to the code shown in step #3 again, we see how these two different events are handled.

```
if ( theFullPosDlg.DoModal( ) == IDOK ) {
    // get the new data back from the dialog
    // note that any validity testing logic
         // was done inside the dialog box
    m_pPos = theFullPosDlg.m_Pos;
    // Assign result back to the caller (step #6)

    // Now use new bytes now, updating this dialog display
    // We update the displayed data we got back.
    // call or do the "on change" for this item

    // inform user what happened
           theApp.log.AddRtxtReport (enInformative,
                   "The Full Position Vector Data has been updated." true);
}
else {
    // inform user what happened
    theApp.log.AddRtxtReport (enInformative,
                "The Full Position Vector Dialog has been canceled (no change to
                data).", true);
}
```

Note that **step #6** is handled in a similar but reverse fashion to step #2, passing a pointer back with the now updated data. The alternative mode, that of copying bytes, would be as shown below in the *BLOB1.cpp* code.

```
m_pPos = theFullPosDlg.m_Pos;
    // Assign result back to the caller (step #6)
```

Or in the case of a blob we have:

```
// move individual data elements values from the blob
void BLOB1::UnStuffIt(BYTE* pTheOutBytes)
{
    // put the new data back from the dialog
    // to the passed pointer location
    int i;
    for (i=0; i < BLOB1_SIZE; i++) {
        // the dialog data is copied to the pointer
        pTheOutBytes[i] = m_blob1[i];
    }

}
```

And this is preceded by a "pack" call which deals with assembling the bytes in the right order and various byte reversal issues. Here is short fragment dealing with the latitude data elements.

```
// do the m_latitude data element
offset = offset + 2; // adjust from last item
m_blob1[offset+0] = (BYTE) ((m_latitude & 0xFF000000) >> 24);
m_blob1[offset+1] = (BYTE) ((m_latitude & 0x00FF0000) >> 16);
m_blob1[offset+2] = (BYTE) ((m_latitude & 0x0000FF00) >> 8);
m_blob1[offset+3] = (BYTE) ((m_latitude & 0x000000FF) );
```

[122] Pressing the small dismiss/close rectangle at the top-right of the window is the same as pressing the cancel button.

Finally, **step #7** is the code in the dialog box that is run when it shuts down. This is required to clear unused (now dangling) items off the heap and prevent memory leaks. This logic can be part of the "ok" routine calls (as shown below), part of the class destructor, or buried in the control logic as variables are created and populated (as shown in the example `OnBnClickedCheck_rearrate` above).

```
void VehicleStatusDlg::OnOK()
{
    //if lights is not used...
    CButton* theCkBox;
    theCkBox = (CButton*) GetDlgItem (IDC_CHECK_VehicleLights);
    if (theCkBox->GetCheck() == BST_UNCHECKED)
    {
        if (m_pLights != NULL)
            ASN_STRUCT_FREE(asn_DEF_ExteriorLights, m_pLights);
    }

    // if FPV is not used...
    theCkBox = (CButton*) GetDlgItem (IDC_CHECK_FullPositionVect);
    if (theCkBox->GetCheck() == BST_UNCHECKED)
    {
        if (m_pPos != NULL)
            ASN_STRUCT_FREE(asn_DEF_FullPositionVector, m_pPos);
    }
    // etc…
```

The basic rule here is simple; no matter when or by what function it was created, always create new message data on the heap and then destroy it when it is no longer used. If you create messages or data frames on the call stack, then this data will go out of scope when the function in which it was declared goes out of scope (ends). This is only viable if the creator will be alive during the full life of the object and the data is never passed off to other parallel routines.

Creating the final BER message in each dialog is done by simply calling the serialize function yet again, with similar code as shown.

```
asn_enc_rval_t er; // Encoder return value
er = der_encode_to_buffer(&asn_DEF_BasicSafetyMessage,
    theMsg, theBuffer, MAX_MSG_SIZE);
if (er.encoded >= MAX_MSG_SIZE)
          MessageBox("Increase buffer in BSM encode.!",
                "Buffer overflow!",MB_OK+ MB_ICONWARNING);
if(er.encoded == -1) {
    return -1; // Failed to encode the data, report error
    }
return er.encoded;  // Return the number of bytes
```

A few words on reading message content are also provided in the next subsection. A review of the code file *msg.cpp* will provide an overview of how to generically read and proces such messages.

Constraint checking, when required, typically should occur right before the serialize call or right after the deserialize call. Keep in mind that constraint checking on a binary message that failed to correctly deserialize is not typically useful. Similarly, a message structure that fails constraint checking may not encode correctly. The constraint checking process checks only the ASN constraint rules; it does not check other application rules expressed in the English, nor will it detect a failure if the "extension areas" (the ASN code denoted with "…") has additional content. The most common occurrence of this is to have an unknown enumerated value that is beyond those defined in the standard (but allowed by the presence of the "…" mark).

The application code makes use of the extensive ITIS code collection to categorize events in the RSA message dialog. Reading ITIS codes in and organizing them into a sorted and useful array is handled by the file *ITISutil.cpp* which contains a number of routines to automatically download,

install, and use the ITIS codes directly from the ASN source file of the standard. Other routines in this code class handle translations between the ITIS integer codes and the enumerated text phrases. These routines are used by the file *ITISfind.cpp* to allow searching and finding items in the code lists and by the file *ITIScat.cpp* to handle the overall dialog and presentation of the code categories to the user in order to allow selection of which specific codes will be used in a message.

8.2.3 User Interface

The user interface follows the common example source code provided by Microsoft under the trade names **Notepad** and **SUPERPAD**.[123] We have used this example to form a "log" text file editor, on which we have added the functions of the DSRC messages along with extensive notes on what the program is doing. [*Aside*: You can control the overall verbosity of these messages with the options presented in the "preferences" dialog box under the command Help → Preferences.] You can find extensive documentation on this tool in the Microsoft help files that come with the development studio C complier and other development tools that they sell. This is a diverse set of examples on proper programming techniques including OLE and embedded object use. In this project, we do not make use of any of these features other than some minimal use of the rich text editing features. Others can develop this area as they see fit, or the menu items and the dialogs connected to them can easily be removed and added to another program.

8.3 Reusing the Code Base in Your Own Applications

The original ASN1c library was intended for use on UNIX and other embedded systems where a POSIX[124] type environment and support calls were available. In order to use this code in a Microsoft windows programming environment a few basic changes were required. A few other changes were also required to use the Microsoft .NET environment and the common run-time library (CRL) that comes with it.[125] These changes are discussed below. The code base, as it exists today for download, can be simply installed and complied in this environment.

In order to use the code generated by the ASN1c library creation tool, the following steps must be undertaken.

1. Convert each ASN1c file name from *.c to be *.cpp (needed so that the Microsoft compiler can cope).
2. Add the defined term below so that each file wraps itself in simple class.

   ```
   #define __cplusplus
   ```

   Note that some other tools do not do it "auto magically" (MS does, so skip this for Windows).
3. Add a path to the ASN files (which are best kept in their own folder) so they can be found. In the default system this is the folder: \ASN_lib_rxx (xx is the release; we use "36" here).

   Note that the ASN lib uses "xxx" for generated files and <xxx> for its own std (static) files when importing. We are putting BOTH types in the ASN_lib folders. Note also that the ASN lib uses common stock headers such as <stdio.h>. You need not add #include "..\stdafx.h" to the ASN code files.

---

[123] For example see: http://msdn.microsoft.com/en-us/library/ms177543(VS.80).aspx or http://msdn.microsoft.com/en-us/library/aa972157.aspx. Please be aware that various editions of these programs are available for each different version of the framework and compiler tools. We have used a somewhat older version to support a wider variety of developers. You may want to consider the use of the latest version now available for Visual Studio 2010 and .NET Framework 4.

[124] POSIX, for "Portable Operating System Interface" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the UNIX operating system. The "core services" of POSIX are now part of the official ANSI C language and is supported by Windows.

[125] If you are new to merging code from multiple sources into a Microsoft application, this application note may prove useful in dealing with common issues that come up: http://www.codeproject.com/KB/cpp/cppforumfaq.aspx#cl_nostd

4. It is essential to keep the ASN code in an "unmanaged" space because it treats pointer casts in ways that MS and C++ prohibit, hence we need to control this in the compiler process. You can add the line `#pragma unmanaged` to the file *asn_application.h* to remove the code from the CRL rules. Or (much more controllable) you can bracket EVERY import of the ASN lib with it as shown below. This method is preferred because you can then turn it back off.

```
#pragma unmanaged
#include "BasicSafetyMessage.h"  ← for any import from ASN1c
#pragma managed
```

5. You can also use the `/TP` option in the complier to treat ALL files as C++ types (regardless of the file extent), for good measure, but steps 1 to 4 are still required.

6. In the Microsoft .NET world, all enumerated types need to be global (what a poor design!) so you need to add a few unique characters to each instance to make it unique.

7. If you import the PER parts of the code you must also deal with a few additional enumerated types that it uses in the same way.

8. Nested structural declares in the ASN1c library cause problems for the Microsoft compiler which cannot handle this level of complexity. Hence, all nested structural calls have been "flattened" into smaller structures in the headers as needed. Search on the comment "`// DCK mod`" to see examples of this.

All of the above has been completed for the source code, however if you regenerate the ASN library you will need to repeat these steps.

You will notice we have debugging enabled in the ASN1c library with the line:

```
#define EMIT_ASN_DEBUG  1
//DCK added to enable debugging calls
```

This can be turned off as needed. In the application it is used to provide verbose trace calls into the ASN 1c library calls. All such calls are placed in a temporary file which can be displayed in the log (depending on user preference settings) or examined as a file afterwards. Note that this is enabled even in production-release builds and causes the tool to be slower due to the additional processing.

You will also notice that memory heap tracing is enabled with the lines:

```
#define _CRTDBG_MAP_ALLOC
// include Microsoft memory leak detection procedures
```

This can be turned off as needed. It provides (but only when in debug build mode) a dump of various memory allocations during the program run in the "output" window of the debugger environment. It is not present in production-release builds of the tool. In a similar manner, some memory management menu items are present only in debug builds as well.

8.4 Using the Tool to Validate Encodings by Other Tools

The tool allows reading in an encoding developed by other tools, and therefore can be used as a basic means of validating if that encoding is correct (both the rules of ASN and to the rules of the DSRC message set standard). You can also read in such an encoding with other commercially available ASN viewer tools. Having multiple ways to compare and confirm that an encoding is correct facilitates deployment confidence.

All of the message reading code is contained in a class called *msg.cpp* which implements a simple dialog box and processes messages in a do loop. In this dialog window, the user is allowed to select one or more[126] messages, then select what processing is to be done to them. When the "run" button is pressed, the processing is started and the results are displayed on the log screen.

[126] Multiple messages are selected by holding down the SHIFT key or with an Alt-A command to select all files in the current folder; i.e. normal Windows file select commands are used.

The dialog seeks to read "*.ber" files which are binary data files encoded in the normal ASN BER-DER way.

The message processing can be controlled in a wide variety of ways. The nine steps performed for each message/file in turn are:

1. Basic Validations and Inspection
   Look over the message for obvious flaws and gross encoding errors
2. Print the Input File as a Byte Stream
   Display the binary data in a hex string to the log
3. Decode Message Type to Use
   Determine what type of message this is, and therefore how to decode it
4. Decode ASN Into a Memory Structure
   Decode the message into its component parts and build a memory structure
5. Validate that ASN Constraints Are Met
   Compare the limits for each element entry to confirm they are in bounds
6. Perform Limited Non-ASN Rule Checks
   Perform some simple comparisons (subject to future expansion) to check any rules not expressed in ASN which the message must also conform with
7. Perform "Application" Non-ASN Rule Checks
   Perform some simple comparisons (subject to future expansion) to check any "application" or use case rules not expressed in ASN which the message must also conform to be legal for that application[127]
8. Print the Structure in Its XML Form
   Display the decoded structure data in an XML string to the log
9. Gather Results of Above Steps and Display
   Display results of the run in a report summary form

In addition to the tool's intrinsic ability to read messages, a set of over 1200 "test" BSMs is provided in three sets of 400 messages each. And a few well-formed messages of each defined type are also provided. These can be used in the tool or in developer applications to confirm the ability to read messages correctly. Typically, the content used is valid but random in nature.[128]

8.5 How to Install the Tools for Use or for Programming

8.5.1 For Use

Obtain a copy of the application file (the file is called DSRCGuide.exe) and place it wherever you wish on your PC. The location **/programfiles/DSRC** would be a typical location. There is no installer script, but the program is often found with accompanying support files in a .zip format. Unpack these files and place them in the same folder. None of these files is required (they will be downloaded from the web when first needed if not present) but they can be helpful. The various "test" messages are often sent this way as well.

[127] This is not yet implemented; it can be added once application rules become apparent. The BSM safety subcommittee is now developing such rules as part of the "profile" for BSM use in V2V applications. Other messages and other subcommittees are expected to also develop such rules over time.

[128] Random in that the data contents, while completely valid, have no coherent relationship from one message to another as would be expected if a stream of messages were to come from a single vehicle proceeding along in time and space.

As the program runs, it downloads any needed files and keeps a copy of them. The program does not use the machine's internet connection for any other use without your express command[129] - the only "automatic" use is for these downloads. The tool can send email using your default email system if you enable that function.[130] The tool can launch a browser window which opens a web site with a record of the SAE DSRC and IEEE DSRC committee email logs.

Whenever the program runs, it creates two folders in that directory (if they do not exist from a prior run). The first of these is the **/msgs** folder where all created message content is stored by default. The second is the **/support** folder where various downloaded files (such as the ITIS codes and the DSRC source listings) are kept. The program presumes it has privileges to write and read these folders at will.

Notes for virus detection tools: The program is "well behaved" in this regard, but it does write to the registry when first opened (to register its file type and to store various preferences). It will attempt to run other programs you have registered to allow you to look at files (text files, BER files, .pdf, Word files, etc.) using whatever tools your PC uses; this may cause an alert to appear with some detection tools. If the machine has a tool registered to open BER type files, it will be used to view these files.

8.5.2 For Programming

The source code is delivered in a single .zip file of around 40 megs in size with about 800 source files.[131] Install the .zip file wherever you prefer to develop code. Open the developer studio tool, and in the project properties dialog window, correct the included file paths as needed to reflect your installation location. Compile the project and begin. The object files have been removed from this .zip file in order to save space. The first time you recompile the project, these files will be recreated (the resulting directory size is about 300 megs). This may take some time, as several hundred files are being created.[132] You will initially observe a great many "warning" messages from the ASN1c files due to its rather cavalier use of pointer casts and enumerated values in conditional expressions. These may be safely ignored. You will also see a number of "unsafe" warnings for simple UNIX/C routines that in the Microsoft environment have been the subject of possible security[133] leaks. These also may be ignored.

8.6 The CommView Utility

All the UDP packet listing and visual network exchanges presented in this guide have been captured with the use of the CommView network packet analyzer. This product is available from TamoSoft (see http://www.tamos.com/). It is strongly suggested that developers equip themselves with a similar tool before attempting to debug DSRC or any other packet based message type protocol.

---

[129] This is true as shipped. If, however, you enable the "chat" message exchange system you may then have local IP traffic from nearby nodes sending and receiving DSRC message traffic to your machine.

[130] This is true only if you enable or register it in the source code. A basic MAPI-MIME email system is provided to allow creating a code support environment; if you do not have MAPI it will appear disabled. You must also register the DLL (either compile it or double click the source code file `Email/Email.dll`) before using it for the first time.

[131] Another 1000~1500 additional text BER files may be present depending on the release.

[132] Approximately 14 minutes is required on a 1.6G P4 system.

[133] A typical example would be the message*: warning C4996: 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead.* The function strcpy_s() is presumed not subject to buffer overrun exploits. In an embedded system this is never a concern, so we have made no effort to correct such errors here.

Annex A – Answers to Selected Questions

1. Q For whom is this Implementation Guide intended?

   Answer: The Guide was developed with the needs of actual implementers and developers in mind, those who need details of how to construct these messages. It covers many details of how the messages are encoded in the language ASN and how a developer can utilize both the message set itself and the encoding provided by ASN BER-DER to create messages that comply with the "over the wire" formats defined by the SAE standard. It mentions, but does not dwell on, a host of other design requirements that must be met in any actual deployed systems.

2. Q Does the DSRC Message Set standard (SAE J2735) provide the requirements data needed to develop a complete Vehicle Safety Application?

   Answer: No, it does not provide what might be called a "full set of product requirements;" rather, it defines in exact detail the messages that are exchanged between various OBUs and RSUs in such a system. As such, it is an interface document and forms a part of the system specification process. Additional specifications and performance requirements are expected to be found in other documents.

3. Q What is the difference between a "data element" and a "data frame" or a "data concept" in these documents?

   Answer: In ITS standards (of which the SAE DSRC message set is one example) the term "data element" refers to an atomic data definition that has no further inner content or construction. The term "data frame" refers to a data definition that does have one or more additional internal definitions that make it up (i.e., complex content). The term "data concept" refers to either of these, or to a message (which is simply a top level data frame that is exchanged with others) in a generic way. Message sets are typically made up of collections of all of these to produce the complete definition of what a message consists of down to the specific bit level.

4. Q Where does one go for the rest of the system's performance specifications needed to build a device that would confirm to these messages?

   Answer: At this time these documents are still in development. The J2735 annex sections reflect (in an informative way) the current best thinking of the industry for various application needs. It is expected that the SAE will soon begin work on developing a more formal set of system level performance specifications for selected applications.

5. Q How does the encoded DER message relate to the payload data carried by the WSM or UDP formats in the lower layers?

   Answer: The encoded DER message (starting with the value `0x30` and concluding with the last byte of the last data element) is the payload of the UDP or WSM message; there are no other layers or intermediate bytes involved. In creating a message to be passed to the lower layers, it may be necessary to provide some additional data (so-called management plane information such as a message priority) but none of that data is placed into the payload or causes the actual DER encoding to change. The lower layers return this same message, intact, as the payload when a message is received.

6. Q What is a PDU (a term commonly used in other communications literature) and how does that relate to these messages?

Answer: The term "Protocol Data Unit" is used in many communications systems to refer to content provided by one layer to another. In some contexts it refers to a message or part of a dialog exchange (as in SNMP use). For the purposes of the DSRC effort, a PDU is the same as the SAE DSRC message itself and consists of a complete well-formed ASN BER-DER encoded message or a well-formed HTML-XML-UTF8 encoded message. In the WSM protocol, only the ASN variant is used.

7. Q What is the difference between "ASN syntax" and "ASN encoding" and why do we care?

Answer: *ASN Syntax* refers to the language of ASN used to define the messages themselves (called an ASN specification). As part of an international treaty, all US ITS standards are expressed in ASN syntax. This language is well established and used in many efforts in the communications industry. Over the past decades, many tools and processes have been developed to support the ASN process flow. By contrast, an *ASN encoding* is the choice of method (typically of one specific flavor from among several) used to translate the ASN specification into an *over the wire* encoding format for transmission to different types of end users. Various encodings can be chosen (in the DSRC message set we use a flavor that is called DER) and each has its proponents and its risks and benefits. In many ITS standards, only the ASN syntax is used (no encoding is presented or recommended). Once an encoding is chosen, the syntax can be changed in subtle ways to take best advantage of that encoding style; this has been done in the DSRC work as well. In order to implement DSRC messages, one must become proficient in basic ASN syntax and also in many aspects of the BER-DER encoding rules, although the ASN library manages many of the details.

8. Q How is the "ASN library" created and where does one get it? How does this differ from the "ASN specification" that the standard provides?

Answer: The ASN library is the output of the ASN tool used. Using the ASN specification (as published in the standard and perhaps with any additional changes made locally) the ASN tool produces the ASN library that will be used with that ASN to produce ASN encodings. The library is in fact a set of many separate files that collectively implement the ASN encoding and decoding of the standard. The library is then used by any application (along with the additional logic of that application) that wishes to manage the messages. The ASN tool is typically used only to produce a new copy of the library when changes are made; the resulting library is then linked to the final application being developed. In most ASN tools, the library files are made up of generic files based on templates that are named and customized to fit the names of the declarative types found in the ASN specification. See section 6. for additional details of how this works with the ASN1c tool we are using.

9. Q What is the difference between "BER" and "DER" or "CER"" and why do we care?

Answer: These are all three different types of ASN encodings (PER is also a well known encoding but not used in DSRC work). The BER (Basic Encoding Rules) were the first rule set developed by the ASN committee for encoding. Various ambiguities found in the rules resulted in the two further efforts, CER (Canonical Encoding Rules) and DER (Distinguished Encoding Rules) to resolve issues that early deployments discovered. Both CER and DER are considered subsets of the BER rule set. You will often see the term BER when CER or DER would in fact be more precise to use. All of these methods encode on a *byte-wise* boundary, but they differ in how arbitrary length objects are handled (which do not occur in DSRC); see

section 6. for more details. In the DSRC message set, we encode all messages in either the BER-DER form (taken directly from the ASN specification) or in an XML form (as defined by the XSD schema present in the published standard).

10. Q Is the PSID value used in the actual DSRC message anywhere? If not, then where is it used and how?

Answer: No, it is only used by the management plan of the lower layers. In that context it is used to group related sets of messages into application categories. It does not appear in the actual message anywhere. The `DSRCMsgID` is the data element used to differentiate one message type from another.

11. Q Is the T-L-V triplet structure pattern used everywhere in ASN encoding?

Answer: Yes, all data concepts, including those that include other data concepts follow this simple pattern. By parsing over the *tag-length-value* groups, you can decode the message, even without the underlying ASN specification. In general, *tags* are numbered sequentially from zero (the first is zero, the second is one, the next is two, etc.) and contain a few additional information flags in the upper nibble (see the section 6. for details). As a result, the tag byte values are found to be sequences like `0x80`, `0x81`, etc. The *length* is typically a single byte (but sign extension rules allow for values longer than 128) and denotes how many more bytes follow. Finally. the *value* (which may be nonexistent if the length is zero) follows. This value may in fact contain other encodings (which follow the same rules) or be simple content. All DSRC messages start with a sequence (a nested construction) that follows these same rules. Therefore the first byte (the outermost tag) is always `0x30`. The second byte is the count of the remaining size of the message. The third byte is then the start of the value for the sequence and also the first tag for the first message data element item. These rules are uniformly applied to the entire message and can be used to decode any ASN that is encoded in the DER style. Section 6. provides a more in-depth description of these rules.

12. Q Why are some values such as *LaneNumber* encoded as an "`OCTET STRING`" rather than as a simple integer as one might expect?

Answer: Because all integers in ASN encoding are considered "signed" and in a case like this (when an unsigned value is required) we need to "cheat" the ASN rules to a slight degree to ensure that we can send and use an 8 bit quantity in a single value byte. Were we to encode this as an unsigned integer [`INTEGER (0.255)`] then the ASN encoding over the wire would require two bytes for the value when the value was in the range 128 to 255 to preserve the sign bit (and only one byte the rest of the time). We do not want to waste bandwidth bytes in this way and so we encode the data concept as an octet string to preserve the range we desire. This occurs in many places in the ASN to save transmission bandwidth.

13. Q Why does the size of the values transmitted vary with the actual content being sent?

Answer: The BER-DER encoding style always uses the fewest number of whole bytes needed to represent the value when sending. So if a value of zero is to be sent, it occupies 8 bits, while if a value of 4096 (13 bits) is sent it will occupy two bytes (16 bits). In addition, all values are treated as signed (there is no such thing as an unsigned value in DER) and this requires that the top-most bit always be interpreted as a sign bit. An often unanticipated side effect of this is that a value like 255 or `Integer (0..255)` requires two full bytes for the value when the value sent is between 128 and 255.

14. Q How can I determine the number of actual value bytes a given instance of an integer contains in a message?

Answer: The length value of the T-L-V triplet tells you. Note that the size or range can also tell you what to expect. If the element is defined as `Octet String (SIZE(3))` then the length is always 3 bytes; if the definition was `INTEGER (0..512)` then the length count could be one or two bytes depending on the actual value. When decoding a value, this rarely comes up because the ASN library performs the decoding for you, extracting the right number of bytes from the stream, interpreting them, and creating a valid variable. So as a programmer, you are handed an integer or a octet string of known byte size and need never deal with the actual length itself. Another side effect of this is that it is often imposable to directly decode a value out of "the middle" of the ASN itself; it is necessary to sequentially decode up to the value in order to detect the variations of lengths and values that may precede it.

15. Q How is the content of an `OCTET STRING` defined "inside" the octet and how can I be sure I have implemented this correctly?

Answer: From a strict ASN-only point of view, the contents of an `OCTET STRING` are not defined by the ASN layer; they are defined by the comments that are associated with the `OCTET STRING` (i.e., the rest of the standard in the case of the DSRC work). In order to be valid (according to the standard) these additional rules must be followed, but the ASN library is not able to detect or enforce them. Therefore the responsibility falls on the device builder to ensure that they have followed all the additional content rules found in the ASN comments and in the description text of the document. This is generally demonstrated by use or comparison with known correct message implementations.

16. Q Is a message that is a valid ASN message (according to the details of the ASN specification of the standard) always a valid DSRC message?

Answer: No, it may not be. It would only be considered a valid DSRC message if all the additional rules concerning its content construction were also followed. These rules are found in the ASN comments for many items (especially for the internal contents of octet string blobs) as well as in the use and remarks sections of the standard. Such additional information is considered normative to the message set even though it is not expressed in an ASN or XML syntax. In addition, specific applications can and will establish additional rules for the encoding of data and the exchange of messages they use in the future. This is expected to be the primary topic of the application performance specification work which is not yet started.

17. Q Is the "Length" in the T-L-V always contained in a single byte of data?

Answer: No. While almost all lengths found in the T-L-V triplets of DSRC standard have values of less than 128 bytes, this need not always be true. Like the rules for signed values in the value expression, the length (and also the tag) reserves the most significant bit to indicate that an additional one or more bytes is required.

18. Q Is the "Tag" in the T-L-V always contained in a single byte of data? How does this work for "local" defined data?

Answer: No. While almost all tags found in the T-L-V triplets of the DSRC standard have values that fit in 5 bits, this need not always be true. Like the rules for signed values in the value expression, the tag (and also the length) reserves some selected MSBs to indicate that an

additional one or more bytes will be required. This can be seen in the range of local tags (which are assigned values from 128 to 255) and these tags always require two bytes to send.

19. Q When a data frame defined as an `OCTET STRING (SIZE (X..Y))` is made up of other data elements defined in the standard but packed together, does the "sign extension" rules of ASN still apply to its contents?

Answer: No it does not. This is the one time it does not. The content of `OCTET STRING` definitions is defined by its additional comments on its construction, typically found in the ASN comments of that expression. Typically, data concepts (which are well defined by other entries in the document) are simply packed (MSB byte and MSB bit first, i.e., *big endian*) into the octet object as a blob. Note that this type of encoding more or less cheats the ability of the ASN library to decode the inner content and that inevitably the application layer must further process the blob to extract the content and determine if it is valid. The rationale for doing this type of encoding is to save additional bandwidth space by removing the some of the tagging that would otherwise be present.

20. Q Why is it that in the C code examples, some objects are referenced by indirect pointers while other objects are not?

Answer: This "feature" is peculiar to the ASN library we are using (the ASN1c tool) but is typically found in tools from other vendors as well. When a data concept is shown in the enclosing .h structure file as being an indirect pointer, it indicates that that element is `OPTIONAL` in the defining ASN specification. If the pointer is in fact pointing to something, then the data concept is present in the message (and a `NULL` value indicates its absence). Note that the management of the storage space required by the object pointed to is then the responsibility of the programmer (typically allocated with `malloc()` and destroyed when needed). If a data concept is required in a message, then a pointer is not used (the object is defined explicitly along with any storage as part of the .h description). When examining a recovered message in memory, a null pointer indicates the absence of the item in question.

21. Q What is this "list" structure that the ASN library has created when there is a *sequence of* in the ASN code?

Answer: This particular ASN tool (ASN1c) invents a structure called *list* to hold the members of any instance of a `SEQUENCE OF` when in memory. Recall that a `SEQUENCE OF` is a list of the same element type repeated. Each inner element is tagged in the normal T-L-V way, but the entire structure is placed into the list object when it is instantiated into memory. You will find that the ASN library provides helper functions to add and remove items from the list (internally it is implemented as linked list in memory) as well as to count the members in it. In the DSRC message set, the `SEQUENCE OF` is typically used to hold complex data frames; any occurrence which would hold simple values (such as integers) is typically expressed as an octet string to reduce the intermediate tagging and byte count.

22. Q Can a valid message contain bad data and still be valid?

Answer: Yes, of course. Consider a basic safety message (BSM) where the latitude value was nonsensical but well-formed (any 4 byte integer value within the range allowed). Such a message is valid from an ASN and DSRC specification point of view, even if the value of latitude made no sense.

23. Q What do you do when you do not have the data required by a message?

Answer: Almost all data element definitions provide instructions for what value to send when there is no data present or when the data is not available (these two semantic use cases differ in some cases). In other cases the use of the data element is itself `OPTIONAL` and is simply not sent.

24. Q Can the length of a value in the T-L-V be zero and when does this occur?

Answer: Yes it can, and when this occurs there is no *value* present, you simply skip to the next T-L-V triplet. This occurs when a structure is defined but in fact has no actual content, such as when a `SEQUENCE` is composed entirely of `OPTIONAL` elements and all of them are absent. In such a case, we have an empty structure and the *length* becomes the value zero and the *value* is not transmitted (as before, it is rounded off to the smallest number of whole bytes needed to represent it, which is zero in this case).

25. Q If I add "local data" to my message is it still a valid SAE DSRC message?

Answer: Yes, as long as it was added properly (using the allotted range of local tags). Such a message is considered valid, and other conformant deployments should be able to read the data concepts in your message that are specified by the original standard to which they were built, while skipping over the additional content you have added.

26. Q How do I know which optional elements are needed or even required in a specific application or message instance?

Answer: First, it is important to recall that the term `OPTIONAL` in the message definitions refers to other possible variations in the structure of the message; it does not refer to which of those structures may or may not be present in a particular use case of the message. To determine if a specific element is needed (required) in a specific message used in a specific application, you must consult the various performance requirements for that application. For example, the *DE_EventFlags* data concept is `OPTIONAL` in Part II of the BSM, but it is obviously required when applications wish to exchange vehicle event information, hence it becomes a required element at those times. If a specific element is not required to be sent in the application of interest, you may remove it from the ASN specification; however, this is not recommended for receiving devices which still need to be prepared to process messages coming from others where any possible such content is present.

27. Q I have new content or a refinement to an existing message that I want to make. How do I proceed?

Answer: This can be easily accommodated by using the local data method to create a tag for your unique item (in the range 128~255) and then adding it to any valid message or data frame where you see the ellipsis symbol (...). You can also reuse existing data concepts in this way. The Implementation Guide, section 7, has additional examples of how this process works. A similar process exists for extending the XML schema.

28. Q When a new revision of one of the messages we have "encapsulated" occurs (such as RTCM Standard 10403.1 for GPS Corrections), how is that reflected in this standard?

Answer: Two basic methods are used when we encapsulate a message from another source. If that message has a revision numbering system, then the revision itself is declared in an

enumerated data element and is then followed by an `OCTET STRING` where the message is contained. If no revision process is created by the authors of the message (which is true in the case of the RTCM messages), then each known variation of the message is given an `OCTET STRING` of its own, typically enclosed in a switch statement to allow choosing which message revision is to be sent. Note that we tend to preserve the entire message in these cases; so often the first few bytes of the encapsulated message also contains its own information about what specific submessage it may be.

29. Q What is the difference between an enumerated value and an integer with a named value?

Answer: In practical fact, both *over the wire* and in the memory footprint supported by the ASN library, nothing. When encoded, ASN treats both like an integer. In the ASN library produced by different vendors some variation in scope is often found; in many tools all such values become global. This may necessitate renaming common enumerated values such as "none" or "unavailable" if they conflict. In such a case, simply append the name of the enclosing data element (i.e., `none -> MUTCDCode_none`)

30. Q How can I tell how many objects are in a *sequence of* statement unless there is an explicit added element that gives the count?

Answer: This is typically not required, but if you need to determine how many items are currently present, use the variable "count" as in `list->count` for the list in question. Lower and upper allowed bounds (constraint ranges) can also be obtained by examining other members of the defining structure. When questions of this nature occur and the tool maker has not taken steps to answer them, examining the actual code in the library is often the best course of action.

31. Q There is a CRC value in the IEEE 1609 and the UDP layers. Why is there also one present in a few of the message definitions and how is it used?

Answer: Each of these layers is using its own CRC value to meet its own needs. In the lower layers a CRC is typically used to confirm the correct receipt of the payload, independent of any additional signing that may be used. In the DSRC message set there is a CRC in a few message types for the purpose of being able to categorize the message contents as being seen before. (These messages are often repeated with the same message content and the CRC is used as a simple way to detect duplicate message receipts.)

32. Q Are data items like *vehicle size* or *vehicle mass* or *vehicle bumper height* expected to ever change over the life of the vehicle for a single OBU?

Answer: Typically these values would be established when the vehicle and OBU were created and never changed over the life of the vehicle. There is some discussion of varying the values slightly (with some dither) in order to better protect vehicle privacy, but at this time no recommended behavior has been established.

33. Q What latency am I allowed to have in the data that is used (for example, if the Controller Area Network bus only allows updating the speed value every 200 mSec, can I still use it in the BSM that comes out every 100 mSec)?

Answer: This is ultimately a performance specification issue and no definitive advice can be given here in the message set documentation. It is expected that the performance will vary across different OBU designs, but no limits have yet been set. Obviously less delay is better, but how this relates to a suitable system performance allocation budget is unknown at this time.

34. Q How many breadcrumbs, probe message snapshots, path predictions, etc., does a device need to be able to generate or to accept in a message?

Answer: The answer to this varies with both design decisions made for the device (out of scope for the message set) and by what the message set itself states as possible in each case. In term of *sending* messages, the device designer may decide to implement in such a way that it can only send a subset of what is defined in the message set for various reasons. In terms of *receiving* messages, any device which cannot handle the longest defined instance of a message is likely to have problems with type acceptance. It is typical in type acceptance testing to send content which demonstrates the correct handling of messages with the largest lengths and values. The largest allowed message can easily be determined by looking at the size and count range restrictions which each data concept in the message has as part of its ASN. For example, the `ProbeVehicleData` message allows up to 32 instances of the `Snapshot` element with the ASN line:

```
snapshots SEQUENCE (SIZE(1..32)) OF Snapshot
```

35. Q Do we have to follow the ASN and/or XML formats defined by the standard in all respects? When can we ignore this or do it differently?

Answer: No. Because different ASN specifications or XML schema can still produce the same *over the wire* message encoding, this is allowed to occur. See question #37 as well. Regardless of any changes made, the over the air format must remain as that defined by the standard.

36. Q The standard itself never used any floating point representation anywhere, but we use it internally in our design. Is there any level of precision we need to achieve in converting the formats for use in messages?

Answer: The message set standard does not define how to internally design and build a system, it only defines the message content (and to some degree the message content accuracy) that is to be exchanged. If the representation used for a floating point value is correctly converted to the fixed representation described by the message set for that element, then that is the extent of the conformance criteria set in the message set. However, other documents, such as systems performance specifications, may establish accuracy and precision requirements beyond this that need to be followed.

37. Q Do we have to follow the ASN specification of the standard exactly? Can we change namespaces and other “internal” representation to suit a specific ASN tool we prefer?

Answer: It is acceptable to change the ASN specification from that found in the standard and used in a deployment as long as the resulting changes do not affect the “over the air” format of the resulting messages (by which conformance with the standard is to be judged). Various ASN tools may require this in small ways (such as not supporting external namespaces in modules). Of course ASN that is not to be implement may be removed. Names of ASN structures may be changed when needed as long as the final structure transmitted still matches the original specification. Additional enumerated types and values and named integers can be created. When in doubt about a specific change, the critical issue to be considered is *does the resulting ASN encoding over the wire differ*. If there is no change in the encoding bit stream, then the ASN specification change is acceptable from a conformance point of view.

38. Q Do we have to implement the entire ITIS table in order to use it?

Answer: No, however doing so is typical in order to be able encode and decode all possible message content. There are valid design needs for low-end devices that will only implement a small subset of the ITIS messages (say it only encoded a few "canned" message content items), but in general any device that is decoding a message with the ITIS codes needs a full and complete ITIS set. To be fully compliant with the standard, your devices need to be able to operate in an environment with any valid ITIS content, even if some codes are not decoded and known (presumably they are discarded). Mapping ITIS codes to display icons or other types of display media is to be encouraged, but there are no requirements for how this is done by the message set standard.

39. Q We do not plan to use or support some of the messages in the standard in our application. Do we have to implement all of these to be compliant?

Answer: No, only the messages required for your supported applications need be implemented, not the whole of the standard. It is also acceptable if the ASN and library you use reflects this subset (you may remove unused ASN from the specification used before creating your ASN library). To be fully compliant with the standard, your device does need to be able to operate in an environment with other messages you do not use (without harm, presumably they are discarded) and this includes any received messages that contain additional message elements as per the tagging details of sections 11.4 and 11.5 for new content.

40. Q Sections 11.4 and 11.5 of the standard state some additional ASN encoding and decoding rules that are required to be "future proof" (forward-compatible with message content not yet defined) in the standard. Do we have to implement these?

Answer: These two sections outline the additional requirements that a device has to be able implement in order to parse over any message content defined after it is deployed without error. Conformant devices must support this process so that they can deal with the presence of messages that may contain additional data of which they are not aware.

41. Q How is the XML schema presented in the standard expected to be used when DER is the more typical encoding for all pubic safety messages?

Answer: The XML schema is to be used when encoding message into XML. This will occur when the WSM protocol is not used, such as when IP is used on a service channel. This generally occurs for less time-sensitive messages. Note that some message types might be encoded in BER-DER for use over the WSM channel for one message, and encoded as XML for transmission over IP and a service channel another time, depending on the specific content that it contains. The XML schema is also expected to be used when sending these messages between other intermediate points over wire line services (such as to or from an RSU for transmission).

42. Q Are the annex sections which appear in the standard *informative* or *normative* to device developers?

Answer: At this time all the annexes are considered informative.

Annex B – Files Produced by the ASN1c Library

Given the ASN source code listed in Annex C the following set of files is returned by the ASN1c compiler tool. Taken as a whole, these implement the entire SAE J2735 message set. The first table of 58 files represent static files used by the tool to provide supporting functions as needed. The second table of 498 files represent the type definitions created by the DSRC source code, each defined type as two files.

Table 15 – Static Support Files

Source Files	Header Files
BOOLEAN.c	BOOLEAN.h
INTEGER.c	INTEGER.h
BIT_STRING.cc	BIT_STRING.h
OCTET_STRING.c	OCTET_STRING.h
IA5String.c	IA5String.h
NativeEnumerated.c	NativeEnumerated.h
NativeInteger.c	NativeInteger.h
	asn_application.h
	asn_codecs.h
	asn_internal.h
	asn_system.h
asn_codecs_prim.c	asn_codecs_prim.h
asn_SEQUENCE_OF.c	asn_SEQUENCE_OF.h
asn_SET_OF.c	asn_SET_OF.h
constraints.c	constraints.h
constr_CHOICE.cc	constr_CHOICE.h
constr_SEQUENCE.c	constr_SEQUENCE.hh
constr_SEQUENCE_OF.c	constr_SEQUENCE_OF.h
constr_SET_OF.c	constr_SET_OF.h
constr_TYPE.c	constr_TYPE.h
ber_tlv_length.c	ber_tlv_length.h
ber_tlv_tag.c	ber_tlv_tag.h
ber_decoder.c	ber_decoder.h
der_encoder.c	der_encoder.h
xer_support.c	xer_support.h
xer_decoder.c	xer_decoder.h
xer_encoder.c	xer_encoder.h
per_support.c	per_support.h
per_decoder.c	per_decoder.h
per_encoder.c	per_encoder.h
per_opentype.c	per_opentype.h

Table 16 – Files Produced from the DSRC ASN Source Specifications

Source Files	Header Files
AlaCarte.c	AlaCarte.h
BasicSafetyMessage.c	BasicSafetyMessage.h
BasicSafetyMessageVerbose.c	BasicSafetyMessageVerbose.h
CommonSafetyRequest.c	CommonSafetyRequest.h
EmergencyVehicleAlert.c	EmergencyVehicleAlert.h
IntersectionCollision.c	IntersectionCollision.h
MapData.c	MapData.h
NMEA-Corrections.c	NMEA-Corrections.h
ProbeDataManagement.c	ProbeDataManagement.h
ProbeVehicleData.c	ProbeVehicleData.h
RoadSideAlert.c	RoadSideAlert.h
RTCM-Corrections.c	RTCM-Corrections.h
SignalRequestMsg.c	SignalRequestMsg.h
SignalStatusMessage.c	SignalStatusMessage.h
SPAT.c	SPAT.h
TravelerInformation.c	TravelerInformation.h
AccelerationSet4Way.c	AccelerationSet4Way.h
AccelSteerYawRateConfidence.c	AccelSteerYawRateConfidence.h
AllInclusive.c	AllInclusive.h
AntennaOffsetSet.c	AntennaOffsetSet.h
Approach.c	Approach.h
ApproachObject.c	ApproachObject.h
BarrierLane.c	BarrierLane.h
BrakeSystemStatus.c	BrakeSystemStatus.h
BSMblob.c	BSMblob.h
BumperHeights.c	BumperHeights.h
Circle.c	Circle.h
ConfidenceSet.c	ConfidenceSet.h
ConnectsTo.c	ConnectsTo.h
CrosswalkLane.c	CrosswalkLane.h
DataParameters.c	DataParameters.h
DDate.c	DDate.h
DDateTime.c	DDateTime.h
DFullTime.c	DFullTime.h
DMonthDay.c	DMonthDay.h
DTime.c	DTime.h
DYearMonth.c	DYearMonth.h
ExitService.c	ExitService.h
FullPositionVector.c	FullPositionVector.h
GenericSignage.c	GenericSignage.h
Intersection.c	Intersection.h
IntersectionState.c	IntersectionState.h
J1939data.c	J1939data.h
MovementState.c	MovementState.h
NodeList.c	NodeList.h
Offsets.c	Offsets.h
PathHistory.c	PathHistory.h
PathHistoryPointType-01.c	PathHistoryPointType-01.h

Source Files	Header Files
PathHistoryPointType-02.c	PathHistoryPointType-02.h
PathHistoryPointType-03.c	PathHistoryPointType-03.h
PathHistoryPointType-04.c	PathHistoryPointType-04.h
PathHistoryPointType-05.c	PathHistoryPointType-05.h
PathHistoryPointType-06.c	PathHistoryPointType-06.h
PathHistoryPointType-07.c	PathHistoryPointType-07.h
PathHistoryPointType-08.c	PathHistoryPointType-08.h
PathHistoryPointType-09.c	PathHistoryPointType-09.h
PathHistoryPointType-10.c	PathHistoryPointType-10.h
PathPrediction.c	PathPrediction.h
Position3D.c	Position3D.h
PositionalAccuracy.c	PositionalAccuracy.h
PositionConfidenceSet.c	PositionConfidenceSet.h
RegionList.c	RegionList.h
RegionOffsets.c	RegionOffsets.h
RegionPointSet.c	RegionPointSet.h
RoadSignID.c	RoadSignID.h
RTCMHeader.c	RTCMHeader.h
RTCMmsg.c	RTCMmsg.h
RTCMPackage.c	RTCMPackage.h
Sample.c	Sample.h
ShapePointSet.c	ShapePointSet.h
SignalControlZone.c	SignalControlZone.h
SignalRequest.c	SignalRequest.h
Snapshot.c	Snapshot.h
SnapshotDistance.c	SnapshotDistance.h
SnapshotTime.c	SnapshotTime.h
SpecialLane.c	SpecialLane.h
SpeedandHeadingandThrottleConfidence.c	SpeedandHeadingandThrottleConfidence.h
SpeedLimit.c	SpeedLimit.h
TransmissionAndSpeed.c	TransmissionAndSpeed.h
ValidRegion.c	ValidRegion.h
VehicleComputedLane.c	VehicleComputedLane.h
VehicleIdent.c	VehicleIdent.h
VehicleReferenceLane.c	VehicleReferenceLane.h
VehicleSafetyExtension.c	VehicleSafetyExtension.h
VehicleSize.c	VehicleSize.h
VehicleStatus.c	VehicleStatus.h
VehicleStatusRequest.c	VehicleStatusRequest.h
WiperStatus.c	WiperStatus.h
WorkZone.c	WorkZone.h
Acceleration.c	Acceleration.h
AccelerationConfidence.c	AccelerationConfidence.h
AmbientAirPressure.c	AmbientAirPressure.h
AmbientAirTemperature.c	AmbientAirTemperature.h
AntiLockBrakeStatus.c	AntiLockBrakeStatus.h
ApproachNumber.c	ApproachNumber.h
AuxiliaryBrakeStatus.c	AuxiliaryBrakeStatus.h
AxleLocation.c	AxleLocation.h
AxleWeight.c	AxleWeight.h
BarrierAttributes.c	BarrierAttributes.h

Source Files	Header Files
BrakeAppliedPressure.c	BrakeAppliedPressure.h
BrakeAppliedStatus.c	BrakeAppliedStatus.h
BrakeBoostApplied.c	BrakeBoostApplied.h
BumperHeightFront.c	BumperHeightFront.h
BumperHeightRear.c	BumperHeightRear.h
CargoWeight.c	CargoWeight.h
CodeWord.c	CodeWord.h
CoefficientOfFriction.c	CoefficientOfFriction.h
ColorState.c	ColorState.h
Count.c	Count.h
CrosswalkLaneAttributes.c	CrosswalkLaneAttributes.h
DDay.c	DDay.h
DescriptiveName.c	DescriptiveName.h
DHour.c	DHour.h
DirectionOfUse.c	DirectionOfUse.h
DMinute.c	DMinute.h
DMonth.c	DMonth.h
DOffset.c	DOffset.h
DriveAxleLiftAirPressure.c	DriveAxleLiftAirPressure.h
DriveAxleLocation.c	DriveAxleLocation.h
DriveAxleLubePressure.c	DriveAxleLubePressure.h
DriveAxleTemperature.c	DriveAxleTemperature.h
DrivenLineOffset.c	DrivenLineOffset.h
DrivingWheelAngle.c	DrivingWheelAngle.h
DSecond.c	DSecond.h
DSignalSeconds.c	DSignalSeconds.h
DSRCmsgID.c	DSRCmsgID.h
DYear.c	DYear.h
Elevation.c	Elevation.h
ElevationConfidence.c	ElevationConfidence.h
EmergencyDetails.c	EmergencyDetails.h
EventFlags.c	EventFlags.h
Extent.c	Extent.h
ExteriorLights.c	ExteriorLights.h
FurtherInfoID.c	FurtherInfoID.h
GPSstatus.c	GPSstatus.h
Heading.c	Heading.h
HeadingConfidence.c	HeadingConfidence.h
HeadingSlice.c	HeadingSlice.h
IntersectionID.c	IntersectionID.h
IntersectionStatusObject.c	IntersectionStatusObject.h
LaneCount.c	LaneCount.h
LaneManeuverCode.c	LaneManeuverCode.h
LaneNumber.c	LaneNumber.h
LaneSet.c	LaneSet.h
LaneWidth.c	LaneWidth.h
Latitude.c	Latitude.h
LayerID.c	LayerID.h
LayerType.c	LayerType.h
LightbarInUse.c	LightbarInUse.h
Location-quality.c	Location-quality.h

Source Files	Header Files
Location-tech.c	Location-tech.h
Longitude.c	Longitude.h
MinuteOfTheYear.c	MinuteOfTheYear.h
MinutesDuration.c	MinutesDuration.h
MsgCount.c	MsgCount.h
MsgCRC.c	MsgCRC.h
MultiVehicleResponse.c	MultiVehicleResponse.h
MUTCDCode.c	MUTCDCode.h
NMEA-MsgType.c	NMEA-MsgType.h
NMEA-Payload.c	NMEA-Payload.h
NMEA-Revision.c	NMEA-Revision.h
NTCIPVehicleclass.c	NTCIPVehicleclass.h
ObjectCount.c	ObjectCount.h
ObstacleDirection.c	ObstacleDirection.h
ObstacleDistance.c	ObstacleDistance.h
Payload.c	Payload.h
PayloadData.c	PayloadData.h
PedestrianDetect.c	PedestrianDetect.h
PedestrianSignalState.c	PedestrianSignalState.h
PositionConfidence.c	PositionConfidence.h
PreemptState.c	PreemptState.h
Priority.c	Priority.h
PriorityState.c	PriorityState.h
ProbeSegmentNumber.c	ProbeSegmentNumber.h
RainSensor.c	RainSensor.h
RequestedItem.c	RequestedItem.h
ResponseType.c	ResponseType.h
RTCM-ID.c	RTCM-ID.h
RTCM-Payload.c	RTCM-Payload.h
RTCM-Revision.c	RTCM-Revision.h
SignalLightState.c	SignalLightState.h
SignalReqScheme.c	SignalReqScheme.h
SignalState.c	SignalState.h
SignPrority.c	SignPrority.h
SirenInUse.c	SirenInUse.h
SpecialLaneAttributes.c	SpecialLaneAttributes.h
SpecialSignalState.c	SpecialSignalState.h
Speed.c	Speed.h
SpeedConfidence.c	SpeedConfidence.h
StabilityControlStatus.c	StabilityControlStatus.h
StateConfidence.c	StateConfidence.h
SteeringAxleLubePressure.c	SteeringAxleLubePressure.h
SteeringAxleTemperature.c	SteeringAxleTemperature.h
SteeringWheelAngle.c	SteeringWheelAngle.h
SteeringWheelAngleConfidence.c	SteeringWheelAngleConfidence.h
SteeringWheelAngleRateOfChange.c	SteeringWheelAngleRateOfChange.h
SunSensor.c	SunSensor.h
TemporaryID.c	TemporaryID.h
TermDistance.c	TermDistance.h
TermTime.c	TermTime.h
ThrottleConfidence.c	ThrottleConfidence.h

Source Files	Header Files
ThrottlePosition.c	ThrottlePosition.h
TimeConfidence.c	TimeConfidence.h
TimeMark.c	TimeMark.h
TireLeakageRate.c	TireLeakageRate.h
TireLocation.c	TireLocation.h
TirePressure.c	TirePressure.h
TirePressureThresholdDetection.c	TirePressureThresholdDetection.h
TireTemp.c	TireTemp.h
TractionControlState.c	TractionControlState.h
TrailerWeight.c	TrailerWeight.h
TransitPreEmptionRequest.c	TransitPreEmptionRequest.h
TransitStatus.c	TransitStatus.h
TransmissionState.c	TransmissionState.h
TravelerInfoType.c	TravelerInfoType.h
TxTime.c	TxTime.h
UniqueMSGID.c	UniqueMSGID.h
URL-Base.c	URL-Base.h
URL-Link.c	URL-Link.h
URL-Short.c	URL-Short.h
VehicleHeight.c	VehicleHeight.h
VehicleLaneAttributes.c	VehicleLaneAttributes.h
VehicleLength.c	VehicleLength.h
VehicleMass.c	VehicleMass.h
VehicleRequestStatus.c	VehicleRequestStatus.h
VehicleStatusDeviceTypeTag.c	VehicleStatusDeviceTypeTag.h
VehicleType.c	VehicleType.h
VehicleWidth.c	VehicleWidth.h
VerticalAcceleration.c	VerticalAcceleration.h
VerticalAccelerationThreshold.c	VerticalAccelerationThreshold.h
VINstring.c	VINstring.h
WheelEndElectFault.c	WheelEndElectFault.h
WheelSensorStatus.c	WheelSensorStatus.h
WiperRate.c	WiperRate.h
WiperStatusFront.c	WiperStatusFront.h
WiperStatusRear.c	WiperStatusRear.h
YawRate.c	YawRate.h
YawRateConfidence.c	YawRateConfidence.h
EssMobileFriction.c	EssMobileFriction.h
EssPrecipRate.c	EssPrecipRate.h
EssPrecipSituation.c	EssPrecipSituation.h
EssPrecipYesNo.c	EssPrecipYesNo.h
EssSolarRadiation.c	EssSolarRadiation.h
IncidentResponseEquipment.c	IncidentResponseEquipment.h
ITIScodes.c	ITIScodes.h
ITIScodesAndText.c	ITIScodesAndText.h
ITIStext.c	ITIStext.h
ResponderGroupAffected.c	ResponderGroupAffected.h
VehicleGroupAffected.c	VehicleGroupAffected.h

Annex C – The ASN.1 and XML Source Code

```
--
-- Output of registry data into file:
-- DSRC_R36_Source.ASN
-- in format need to operate on the ASN source files.
--
-- Run on Mini-Edit Version 3.1.500
-- From file: \DSRC_36\Dsrc_rev036.ITS
-- Last Changed:  [Mod: 10/28/2009 3:08:03 PM]
-- This export was created on 11/11/2009  at 1:15:00 PM
-- web: http://www.itsware.net/itsschemas/DSRC/DSRC-03-00-36/
-- ###################################################
--
--
-- Run this file with a line like:
-- asn1 source.txt -errorfile errs.txt -noun
--
-- The local module consisting of DEs / DFs and MSGs
DSRC DEFINITIONS AUTOMATIC TAGS::= BEGIN

-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--
-- Start of entries from table Dialogs...
-- This table typicaly contains dialog and operational exchange entries.
-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--

-- Table contains no entries.

-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--
-- Start of entries from table Messages...
-- This table typicaly contains message entries.
-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--


-- MSG_A_la_Carte  (ACM) (Desc Name) Record 1
AlaCarte ::=  SEQUENCE {
   msgID         DSRCmsgID,
                 -- the message type
   data          AllInclusive,
                 -- any possible set of data items here
   crc           MsgCRC OPTIONAL,
   ... -- # LOCAL_CONTENT
   }



-- MSG_BasicSafetyMessage  (BSM) (Desc Name) Record 2
BasicSafetyMessage ::=  SEQUENCE {
   -- Part I
   msgID       DSRCmsgID,                 -- 1 byte

   -- Sent as a single octet blob
   blob1       BSMblob,

   --
   -- The blob consists of the following 38 packed bytes:
   --
   -- msgCnt      MsgCount,              -x- 1 byte
   -- id          TemporaryID,           -x- 4 bytes
   -- secMark     DSecond,               -x- 2 bytes

   -- pos      PositionLocal3D,
     -- lat       Latitude,              -x- 4 bytes
     -- long      Longitude,             -x- 4 bytes
     -- elev      Elevation,             -x- 2 bytes
     -- accuracy  PositionalAccuracy,    -x- 4 bytes

   -- motion   Motion,
     -- speed     TransmissionAndSpeed, -x- 2 bytes
     -- heading   Heading,               -x- 2 byte
     -- angle     SteeringWheelAngle     -x- 1 bytes
     -- accelSet  AccelerationSet4Way,   -x- 7 bytes

   -- control  Control,
   -- brakes      BrakeSystemStatus,     -x- 2 bytes
```

```
   -- basic    VehicleBasic,
   -- size        VehicleSize,          -x- 3 bytes

   -- Part II, sent as required
   -- Part II,
   safetyExt     VehicleSafetyExtension OPTIONAL,
   status        VehicleStatus          OPTIONAL,

   ... -- # LOCAL_CONTENT
   }


-- MSG_BasicSafetyMessage_Verbose (Desc Name) Record 3
BasicSafetyMessageVerbose ::=  SEQUENCE {
   -- Part I, sent at all times
   msgID       DSRCmsgID,            -- App ID value, 1 byte

   msgCnt      MsgCount,             -- 1 byte
   id          TemporaryID,          -- 4 bytes
   secMark     DSecond,              -- 2 bytes
   -- pos      PositionLocal3D,
   lat         Latitude,             -- 4 bytes
   long        Longitude,            -- 4 bytes
   elev        Elevation,            -- 2 bytes
   accuracy    PositionalAccuracy,   -- 4 bytes

   -- motion   Motion,
   speed       TransmissionAndSpeed, -- 2 bytes
   heading     Heading,              -- 2 bytes
   angle       SteeringWheelAngle,   -- 1 bytes
   accelSet    AccelerationSet4Way,  -- 7 bytes

   -- control  Control,
   brakes      BrakeSystemStatus,    -- 2 bytes

   -- basic    VehicleBasic,
   size        VehicleSize,          -- 3 bytes

   -- Part II, sent as required
   -- Part II,
   safetyExt   VehicleSafetyExtension OPTIONAL,
   status      VehicleStatus          OPTIONAL,
   ... -- # LOCAL_CONTENT
   }


-- MSG_CommonSafetyRequest (CSR) (Desc Name) Record 4
CommonSafetyRequest ::=  SEQUENCE {
   msgID       DSRCmsgID,
   msgCnt      MsgCount OPTIONAL,
   id          TemporaryID OPTIONAL,

   -- Note: Uses the same request as probe management
   requests    SEQUENCE (SIZE(1..32)) OF RequestedItem,

   ... -- # LOCAL_CONTENT
   }


-- MSG_EmergencyVehicleAlert (EVA) (Desc Name) Record 5
EmergencyVehicleAlert ::=  SEQUENCE {
   msgID           DSRCmsgID,
   id              TemporaryID OPTIONAL,
   rsaMsg          RoadSideAlert,
                -- the DSRCmsgID inside this
                                -- data frame is set as per the
                                -- RoadSideAlert.  The CRC is
                                -- set to a value of zero.
   responseType    ResponseType                  OPTIONAL,
   details         EmergencyDetails              OPTIONAL,
                   -- Combines these 3 items:
                   -- SirenInUse,
                   -- LightbarInUse,
                   -- MultiVehicleReponse,

   mass            VehicleMass                   OPTIONAL,
   basicType       VehicleType                   OPTIONAL,
                                -- gross size and axle cnt
```

```
   -- type of vehicle and agency when known
   vehicleType     ITIS.VehicleGroupAffected      OPTIONAL,
   responseEquip   ITIS.IncidentResponseEquipment OPTIONAL,
   responderType   ITIS.ResponderGroupAffected    OPTIONAL,
   crc             MsgCRC,
   ... -- # LOCAL_CONTENT
   }


-- MSG_IntersectionCollisionAvoidance  (ICA) (Desc Name) Record 6
IntersectionCollision ::=  SEQUENCE {
   msgID          DSRCmsgID,
   msgCnt         MsgCount,
   id             TemporaryID,
   secMark        DSecond OPTIONAL,
   path           PathHistory,
                  -- a set of recent path histories
   intersetionID  IntersectionID,
                  -- the applicable Intersection, from the MAP-GID
                  -- the best applicable movement, from the MAP-GID
   laneNumber     LaneNumber,
                  -- the best applicable Lane, from the MAP-SPAT-GID
                  -- zero sent if unknown
   eventFlag      EventFlags,
                  -- used to convey vehicle Panic Events,
                  -- Set to indicate "Intersection Violation"
   ... -- # LOCAL_CONTENT
   }


-- MSG_MapData  (MAP) (Desc Name) Record 7
MapData ::=  SEQUENCE {
   msgID           DSRCmsgID,
   msgCnt          MsgCount,
   name            DescriptiveName OPTIONAL,
   layerType       LayerType  OPTIONAL,
   layerID         LayerID  OPTIONAL,
   intersections   SEQUENCE (SIZE(1..32)) OF
                   Intersection OPTIONAL,

   -- other objects may be added at this layer, tbd,
   -- this might become a nested CHOICE statement
   -- roadSegments    SEQUENCE (SIZE(1..32)) OF
   --                 RoadSegments OPTIONAL,
   -- curveSegments   SEQUENCE (SIZE(1..32)) OF
   --                 curveSegments OPTIONAL,

   -- wanted: some type of data frame describing how
   -- the data was determined/processed to go here
   dataParameters   DataParameters OPTIONAL,
   crc              MsgCRC,
   ... -- # LOCAL_CONTENT
   }


-- MSG_NMEA_Corrections  (NMEA) (Desc Name) Record 8
NMEA-Corrections ::=  SEQUENCE {
   msgID     DSRCmsgID,
   rev       NMEA-Revision,
             -- the specific edition of the standard
             -- that is being sent, normally 2.0
   msg       NMEA-MsgType,
             -- the message and sub-message type, as
             -- defined in the revision being used
   -- NOTE as the message type is also in the payload,
   wdCount   INTEGER (0..1023),
             -- a count of bytes to follow
   payload   NMEA-Payload,
   ...
   }


-- MSG_ProbeDataManagement  (PDM) (Desc Name) Record 9
ProbeDataManagement ::=  SEQUENCE {
   msgID               DSRCmsgID,        -- This is a unique message
                                         -- identifier, NOT related to
                                         -- the PSID\PSC
   sample              Sample,           -- identifies vehicle
                                         -- population affected
```

```
   directions           HeadingSlice,
                                       -- Applicable headings/directions
   term CHOICE {
      termtime          TermTime,      -- Terminate management process
                                       -- based on Time-to-Live
      termDistance      TermDistance   -- Terminate management process
                                       -- based on Distance-to-Live
      },
   snapshot CHOICE {
      snapshotTime      SnapshotTime,  -- Collect snapshots based on time
      snapshotDistance  SnapshotDistance  -- Collect snapshots based on Distance
      },
   txInterval           TxTime,        -- Time Interval at which to send snapshots
   cntTthreshold        Count,         -- number of thresholds that will be changed
   dataElements SEQUENCE (SIZE(1..32)) OF
                        VehicleStatusRequest,
                                       -- a data frame and its assoc thresholds

   ...
   }


-- MSG_ProbeVehicleData  (PVD) (Desc Name) Record 10
ProbeVehicleData ::=  SEQUENCE {
   msgID           DSRCmsgID,          -- App ID value, 1 byte
   segNum          ProbeSegmentNumber OPTIONAL,
                                       -- a short term Ident value
                                       -- not used when ident is used
   probeID         VehicleIdent OPTIONAL,
                                       -- ident data for selected
                                       -- types of vehicles
   startVector     FullPositionVector,   -- the space and time of
                                       -- transmission to the RSU
   vehicleType     VehicleType,        -- type of vehicle, 1 byte
   cntSnapshots    Count OPTIONAL,
                                       -- a count of how many snaphots
                                       -- type entries will follow
   snapshots       SEQUENCE (SIZE(1..32)) OF Snapshot,
                                       -- a seq of name-value pairs
                                       -- along with the space and time
                                       -- of the first measurement set
   ... -- # LOCAL_CONTENT
   } -- Est size about 64 bytes plus snapshot sizes (about 12 per)


-- MSG_RoadSideAlert  (RSA) (Desc Name) Record 11
RoadSideAlert ::=  SEQUENCE {
   msgID         DSRCmsgID,
                 -- the message type.
   msgCnt        MsgCount,
   typeEvent     ITIS.ITIScodes,
                 -- a category and an item from that category
                 -- all ITS stds use the same types here
                 -- to explain the type of  the
                 -- alert / danger / hazard involved
                 -- two bytes in length
   description   SEQUENCE (SIZE(1..8)) OF ITIS.ITIScodes OPTIONAL,
                 -- up to eight ITIS code entries to further
                 -- describe the event, give advice, or any
                 -- other ITIS codes
                 -- up to 16 bytes in length
   priority      Priority OPTIONAL,
                 -- the urgency of this message, a relative
                 -- degree of merit compared with other
                 -- similar messages for this type (not other
                 -- message being sent by the device), nor a
                 -- priority of display urgency
                 -- one byte in length
   heading       HeadingSlice  OPTIONAL,
                 -- Applicable headings/direction
   extent        Extent OPTIONAL,
                 -- the spatial distance over which this
                 -- message applies and should be presented
                 -- to the driver
                 -- one byte in length
   positon       FullPositionVector OPTIONAL,
                 -- a compact summary of the position,
                 -- heading, rate of speed, etc of the
                 -- event in question. Including stationary
                 -- and wide area events.
   furtherInfoID FurtherInfoID OPTIONAL,
```

```
                -- a link to any other incident
                -- information data that may be available
                -- in the normal ATIS incident description
                -- or other messages
                -- 1~2 bytes in length
    crc         MsgCRC
    }


-- MSG_RTCM_Corrections  (RTCM) (Desc Name) Record 12
RTCM-Corrections ::=  SEQUENCE {
   msgID     DSRCmsgID,
   msgCnt    MsgCount,
   rev       RTCM-Revision,
             -- the specific edition of the standard
             -- that is being sent

   anchorPoint FullPositionVector OPTIONAL,
   -- precise observer position, if needed

   -- precise ant position and noise data
   rtcmHeader  RTCMHeader,
   -- octets of:
   -- status      GPSstatus
   -- antOffsets  AntennaOffsetSet(x,y,z)

   -- one or more RTCM messages
   rtcmSets    SEQUENCE (SIZE(1..5)) OF RTCMmsg,
   ... -- # LOCAL_CONTENT
   }


-- MSG_SignalRequestMessage  (SRM) (Desc Name) Record 13
SignalRequestMsg ::=  SEQUENCE {
   msgID          DSRCmsgID,
   msgCnt         MsgCount,

   -- Request Data
   request        SignalRequest,
                  -- the specific request to the intersection
                  -- contains IntersectionID, cancel flags,
                  -- requested action, optional lanes data

   timeOfService  DTime  OPTIONAL,
                  -- the time in the near future when service is
                  -- requested to start

   endOfService   DTime  OPTIONAL,
                  -- the time in the near future when service is
                  -- requested to end

   transitStatus  TransitStatus OPTIONAL,
                  -- additional information pertaining
                  -- to transit events

   -- User Data
   vehicleVIN     VehicleIdent  OPTIONAL,
                  -- a set of unique strings to identify the requesting vehicle

   vehicleData    BSMblob,
                  -- current position data about the vehicle

   status         VehicleRequestStatus  OPTIONAL,
                  -- current status data about the vehicle

   ...
   }


-- MSG_SignalStatusMessage  (SSM) (Desc Name) Record 14
SignalStatusMessage ::=  SEQUENCE {
   msgID         DSRCmsgID,
   msgCnt        MsgCount,
   id            IntersectionID,
                 -- this provides a unique mapping to the
                 -- intersection map in question
                 -- which provides complete location
                 -- and approach/move/lane data
                 -- as well as zones for priority/preemption
   status        IntersectionStatusObject,
```

```
                -- general status of the signal controller
   priority     SEQUENCE (SIZE(1..7)) OF SignalState OPTIONAL,
                -- all active priority state data
                -- is found here
   priorityCause VehicleIdent  OPTIONAL,
                -- vehicle that requested
                -- the current priority
   prempt       SEQUENCE (SIZE(1..7)) OF SignalState OPTIONAL,
                -- all active preemption state data
                -- is found here
   preemptCause  VehicleIdent  OPTIONAL,
                -- vehicle that requested
                -- the current preempt
   transitStatus TransitStatus OPTIONAL,
                -- additional information pertaining
                -- to transit event, if that is the active event
   ...
   }


-- MSG_SignalPhaseAndTiming Message  (SPAT) (Desc Name) Record 15
SPAT ::=  SEQUENCE {
   msgID         DSRCmsgID,
   name          DescriptiveName OPTIONAL,
                 -- human readable name for this collection
                 -- to be used only in debug mode

   intersections SEQUENCE (SIZE(1..32)) OF IntersectionState,
                 -- sets of SPAT data (one per intersection)

   ... -- # LOCAL_CONTENT
   }


-- MSG_TravelerInformation Message  (TIM) (Desc Name) Record 16
TravelerInformation ::=  SEQUENCE {
   msgID           DSRCmsgID,
   packetID        UniqueMSGID    OPTIONAL,
   urlB            URL-Base       OPTIONAL,
   dataFrameCount  Count          OPTIONAL,

   dataFrames SEQUENCE (SIZE(1..8)) OF SEQUENCE {

      -- Part I, Frame header
      frameType   TravelerInfoType,  -- (enum, advisory or road sign)
      msgId       CHOICE  {
                     furtherInfoID   FurtherInfoID,
                                     -- links to ATIS msg
                     roadSignID      RoadSignID
                                     -- an ID to other data
                     },
      startYear   DYear OPTIONAL,
                  -- Current year used if missing
      startTime   MinuteOfTheYear,
      duratonTime MinutesDuration,
      priority    SignPrority,

      -- Part II, Applicable Regions of Use
     commonAnchor         Position3D     OPTIONAL,
                          -- a shared anchorpoint
     commonLaneWidth      LaneWidth      OPTIONAL,
                          -- a shared lane width
     commonDirectionality DirectionOfUse OPTIONAL,
                          -- a shared direction of use
     regions  SEQUENCE (SIZE(1..16)) OF ValidRegion,

      -- Part III, Content
      content  CHOICE  {
                  advisory       ITIS.ITIScodesAndText,
                                 -- typical ITIS warnings
                  workZone       WorkZone,
                                 -- work zone signs and directions
                  genericSign    GenericSignage,
                                 -- MUTCD signs and directions
                  speedLimit     SpeedLimit,
                                 -- speed limits and cautions
                  exitService    ExitService
                                 -- roadside avaiable services
                  -- other types may be added in future revisions
                  },  --# UNTAGGED
      url     URL-Short OPTIONAL  -- May link to image or other content
```

```
      },
   crc         MsgCRC,
   ... -- # LOCAL_CONTENT
   }


-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
--
-- Start of entries from table Data_Frames...
-- This table typicaly contains data frame entries.
-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
--


-- DF_AccelerationSet4Way (Desc Name) Record 1
AccelerationSet4Way ::= OCTET STRING (SIZE(7))
   -- composed of the following:
   -- SEQUENCE {
   --    long Acceleration,          -x- Along the Vehicle Longitudinal axis
   --    lat  Acceleration,          -x- Along the Vehicle Lateral axis
   --    vert VerticalAcceleration,  -x- Along the Vehicle Vertical axis
   --    yaw  YawRate
   --    }



-- DF_AccelSteerYawRateConfidence (Desc Name) Record 2
AccelSteerYawRateConfidence ::=  SEQUENCE {
   yawRate             YawRateConfidence,
                       -- 3 bits
   acceleration        AccelerationConfidence,
                       -- 3 bits
   steeringWheelAngle  SteeringWheelAngleConfidence
                       -- 2 bits
   }



-- DF_AllInclusive (Desc Name) Record 3
AllInclusive ::=  SEQUENCE {
   -- Data Frame Items
   item6-1        AccelerationSet4Way                     OPTIONAL,
   item6-2        AccelSteerYawRateConfidence             OPTIONAL,
   -- item6-3     AllInclusive                            OPTIONAL,
   item6-4        AntennaOffsetSet                        OPTIONAL,
   item6-5        Approach                                OPTIONAL,
   item6-6        ApproachObject                          OPTIONAL,
   item6-7        BarrierLane                             OPTIONAL,
   item6-8        BrakeSystemStatus                       OPTIONAL,
   item6-9        BSMblob                                 OPTIONAL,
   item6-10       BumperHeights                           OPTIONAL,
   item6-11       Circle                                  OPTIONAL,
   item6-12       ConfidenceSet                           OPTIONAL,
   item6-13       ConnectsTo                              OPTIONAL,
   item6-14       CrosswalkLane                           OPTIONAL,
   item6-15       DataParameters                          OPTIONAL,
   item6-16       DDate                                   OPTIONAL,
   item6-17       DDateTime                               OPTIONAL,
   item6-18       DFullTime                               OPTIONAL,
   item6-19       DMonthDay                               OPTIONAL,
   item6-20       DTime                                   OPTIONAL,
   item6-21       DYearMonth                              OPTIONAL,
   item6-22       FullPositionVector                      OPTIONAL,
   item6-23       Intersection                            OPTIONAL,
   item6-24       IntersectionState                       OPTIONAL,
   item6-25       ExitService                             OPTIONAL,
   item6-26       GenericSignage                          OPTIONAL,
   item6-27       SpeedLimit                              OPTIONAL,
   item6-28       WorkZone                                OPTIONAL,
   item6-29       J1939data                               OPTIONAL,
   item6-30       MovementState                           OPTIONAL,
   item6-31       NodeList                                OPTIONAL,
   item6-32       Offsets                                 OPTIONAL,
   item6-33       PathHistory                             OPTIONAL,
   item6-34       PathHistoryPointType-01                 OPTIONAL,
   item6-35       PathHistoryPointType-02                 OPTIONAL,
   item6-36       PathHistoryPointType-03                 OPTIONAL,
   item6-37       PathHistoryPointType-04                 OPTIONAL,
   item6-38       PathHistoryPointType-05                 OPTIONAL,
   item6-39       PathHistoryPointType-06                 OPTIONAL,
   item6-40       PathHistoryPointType-07                 OPTIONAL,
   item6-41       PathHistoryPointType-08                 OPTIONAL,
   item6-42       PathHistoryPointType-09                 OPTIONAL,
```

```
item6-43        PathHistoryPointType-10                 OPTIONAL,
item6-44        PathPrediction                          OPTIONAL,
item6-45        Position3D                              OPTIONAL,
item6-46        PositionalAccuracy                      OPTIONAL,
item6-47        PositionConfidenceSet                   OPTIONAL,
item6-48        RegionList                              OPTIONAL,
item6-49        RegionOffsets                           OPTIONAL,
item6-50        RegionPointSet                          OPTIONAL,
item6-51        RoadSignID                              OPTIONAL,
item6-52        RTCMHeader                              OPTIONAL,
item6-53        RTCMmsg                                 OPTIONAL,
item6-54        RTCMPackage                             OPTIONAL,
item6-55        Sample                                  OPTIONAL,
item6-56        ShapePointSet                           OPTIONAL,
item6-57        SignalControlZone                       OPTIONAL,
item6-58        SignalRequest                           OPTIONAL,
item6-59        SnapshotDistance                        OPTIONAL,
item6-60        Snapshot                                OPTIONAL,
item6-61        SnapshotTime                            OPTIONAL,
item6-62        SpecialLane                             OPTIONAL,
item6-63        SpeedandHeadingandThrottleConfidence    OPTIONAL,
item6-64        TransmissionAndSpeed                    OPTIONAL,
item6-65        ValidRegion                             OPTIONAL,
item6-66        VehicleComputedLane                     OPTIONAL,
item6-67        VehicleIdent                            OPTIONAL,
item6-68        VehicleReferenceLane                    OPTIONAL,
item6-69        VehicleSafetyExtension                  OPTIONAL,
item6-70        VehicleSize                             OPTIONAL,
item6-71        VehicleStatusRequest                    OPTIONAL,
item6-72        VehicleStatus                           OPTIONAL,
item6-73        WiperStatus                             OPTIONAL,

-- Data Element Items
item7-1         Acceleration                            OPTIONAL,
item7-2         AccelerationConfidence                  OPTIONAL,
item7-3         AmbientAirPressure                      OPTIONAL,
item7-4         AmbientAirTemperature                   OPTIONAL,
item7-5         AntiLockBrakeStatus                     OPTIONAL,
item7-6         ApproachNumber                          OPTIONAL,
item7-7         AuxiliaryBrakeStatus                    OPTIONAL,
item7-8         BarrierAttributes                       OPTIONAL,
item7-9         BrakeAppliedPressure                    OPTIONAL,
item7-10        BrakeAppliedStatus                      OPTIONAL,
item7-11        BrakeBoostApplied                       OPTIONAL,
item7-12        BumperHeightFront                       OPTIONAL,
item7-13        BumperHeightRear                        OPTIONAL,
item7-14        CodeWord                                OPTIONAL,
item7-15        CoefficientOfFriction                   OPTIONAL,
item7-16        ColorState                              OPTIONAL,
item7-17        Count                                   OPTIONAL,
item7-18        CrosswalkLaneAttributes                 OPTIONAL,
item7-19        DDay                                    OPTIONAL,
item7-20        DescriptiveName                         OPTIONAL,
item7-21        DHour                                   OPTIONAL,
item7-22        DirectionOfUse                          OPTIONAL,
item7-23        DMinute                                 OPTIONAL,
item7-24        DMonth                                  OPTIONAL,
item7-25        DOffset                                 OPTIONAL,
item7-26        DrivenLineOffset                        OPTIONAL,
item7-27        DrivingWheelAngle                       OPTIONAL,
item7-28        DSecond                                 OPTIONAL,
item7-29        DSignalSeconds                          OPTIONAL,
item7-30        DSRCmsgID                               OPTIONAL,
item7-31        DYear                                   OPTIONAL,
item7-32        ElevationConfidence                     OPTIONAL,
item7-33        Elevation                               OPTIONAL,
item7-34        EmergencyDetails                        OPTIONAL,
item7-35        EventFlags                              OPTIONAL,
item7-36        Extent                                  OPTIONAL,
item7-37        ExteriorLights                          OPTIONAL,
item7-38        FurtherInfoID                           OPTIONAL,
item7-39        GPSstatus                               OPTIONAL,
item7-40        HeadingConfidence                       OPTIONAL,
item7-41        Heading                                 OPTIONAL,
item7-42        HeadingSlice                            OPTIONAL,
item7-43        IntersectionStatusObject                OPTIONAL,
item7-44        IntersectionID                          OPTIONAL,
item7-45        AxleLocation                            OPTIONAL,
item7-46        AxleWeight                              OPTIONAL,
item7-47        CargoWeight                             OPTIONAL,
item7-48        DriveAxleLiftAirPressure                OPTIONAL,
item7-49        DriveAxleLocation                       OPTIONAL,
item7-50        DriveAxleLubePressure                   OPTIONAL,
```

```
item7-51      DriveAxleTemperature                OPTIONAL,
item7-52      SteeringAxleLubePressure            OPTIONAL,
item7-53      SteeringAxleTemperature             OPTIONAL,
item7-54      TireLeakageRate                     OPTIONAL,
item7-55      TireLocation                        OPTIONAL,
item7-56      TirePressureThresholdDetection      OPTIONAL,
item7-57      TirePressure                        OPTIONAL,
item7-58      TireTemp                            OPTIONAL,
item7-59      TrailerWeight                       OPTIONAL,
item7-60      WheelEndElectFault                  OPTIONAL,
item7-61      WheelSensorStatus                   OPTIONAL,
item7-62      LaneCount                           OPTIONAL,
item7-63      LaneManeuverCode                    OPTIONAL,
item7-64      LaneNumber                          OPTIONAL,
item7-65      LaneSet                             OPTIONAL,
item7-66      LaneWidth                           OPTIONAL,
item7-67      Latitude                            OPTIONAL,
item7-68      LayerID                             OPTIONAL,
item7-69      LayerType                           OPTIONAL,
item7-70      LightbarInUse                       OPTIONAL,
item7-71      Longitude                           OPTIONAL,
item7-72      Location-quality                    OPTIONAL,
item7-73      Location-tech                       OPTIONAL,
item7-74      MinuteOfTheYear                     OPTIONAL,
item7-75      MinutesDuration                     OPTIONAL,
item7-76      MsgCount                            OPTIONAL,
item7-77      MsgCRC                              OPTIONAL,
item7-78      MultiVehicleResponse                OPTIONAL,
item7-79      MUTCDCode                           OPTIONAL,
item7-80      NMEA-MsgType                        OPTIONAL,
item7-81      NMEA-Payload                        OPTIONAL,
item7-82      NMEA-Revision                       OPTIONAL,
item7-83      NTCIPVehicleclass                   OPTIONAL,
item7-84      ObjectCount                         OPTIONAL,
item7-85      ObstacleDirection                   OPTIONAL,
item7-86      ObstacleDistance                    OPTIONAL,
item7-87      PayloadData                         OPTIONAL,
item7-88      Payload                             OPTIONAL,
item7-89      PedestrianDetect                    OPTIONAL,
item7-90      PedestrianSignalState               OPTIONAL,
item7-91      PositionConfidence                  OPTIONAL,
item7-92      PreemptState                        OPTIONAL,
item7-93      Priority                            OPTIONAL,
item7-94      PriorityState                       OPTIONAL,
item7-95      ProbeSegmentNumber                  OPTIONAL,
item7-96      RainSensor                          OPTIONAL,
item7-97      RequestedItem                       OPTIONAL,
item7-98      ResponseType                        OPTIONAL,
item7-99      RTCM-ID                             OPTIONAL,
item7-100     RTCM-Payload                        OPTIONAL,
item7-101     RTCM-Revision                       OPTIONAL,
item7-102     SignalLightState                    OPTIONAL,
item7-103     SignalReqScheme                     OPTIONAL,
item7-104     SignalState                         OPTIONAL,
item7-105     SignPrority                         OPTIONAL,
item7-106     SirenInUse                          OPTIONAL,
item7-107     SpecialLaneAttributes               OPTIONAL,
item7-108     SpecialSignalState                  OPTIONAL,
item7-109     SpeedConfidence                     OPTIONAL,
item7-110     Speed                               OPTIONAL,
item7-111     StabilityControlStatus              OPTIONAL,
item7-112     StateConfidence                     OPTIONAL,
item7-113     SteeringWheelAngleConfidence        OPTIONAL,
item7-114     SteeringWheelAngleRateOfChange      OPTIONAL,
item7-115     SteeringWheelAngle                  OPTIONAL,
item7-116     SunSensor                           OPTIONAL,
item7-117     TemporaryID                         OPTIONAL,
item7-118     TermDistance                        OPTIONAL,
item7-119     TermTime                            OPTIONAL,
item7-120     ThrottleConfidence                  OPTIONAL,
item7-121     ThrottlePosition                    OPTIONAL,
item7-122     TimeConfidence                      OPTIONAL,
item7-123     TimeMark                            OPTIONAL,
item7-124     TractionControlState                OPTIONAL,
item7-125     TransitPreEmptionRequest            OPTIONAL,
item7-126     TransitStatus                       OPTIONAL,
item7-127     TransmissionState                   OPTIONAL,
item7-128     TxTime                              OPTIONAL,
item7-129     TravelerInfoType                    OPTIONAL,
item7-130     UniqueMSGID                         OPTIONAL,
item7-131     URL-Base                            OPTIONAL,
item7-132     URL-Link                            OPTIONAL,
item7-133     URL-Short                           OPTIONAL,
```

```
   item7-134      VehicleHeight                          OPTIONAL,
   item7-135      VehicleLaneAttributes                  OPTIONAL,
   item7-136      VehicleLength                          OPTIONAL,
   item7-137      VehicleMass                            OPTIONAL,
   item7-138      VehicleRequestStatus                   OPTIONAL,
   item7-139      VehicleStatusDeviceTypeTag             OPTIONAL,
   item7-140      VehicleType                            OPTIONAL,
   item7-141      VehicleWidth                           OPTIONAL,
   item7-142      VerticalAccelerationThreshold          OPTIONAL,
   item7-143      VerticalAcceleration                   OPTIONAL,
   item7-144      VINstring                              OPTIONAL,
   item7-145      WiperRate                              OPTIONAL,
   item7-146      WiperStatusFront                       OPTIONAL,
   item7-147      WiperStatusRear                        OPTIONAL,
   item7-148      YawRateConfidence                      OPTIONAL,
   item7-149      YawRate                                OPTIONAL,

   -- External Items
   item8-1        ITIS.IncidentResponseEquipment         OPTIONAL,
   item8-2        ITIS.ITIStext                          OPTIONAL,
   item8-3        ITIS.ResponderGroupAffected            OPTIONAL,
   item8-4        ITIS.VehicleGroupAffected              OPTIONAL,
   item8-5        ITIS.ITIScodesAndText                  OPTIONAL,
   item8-6        NTCIP.EssMobileFriction                OPTIONAL,
   item8-7        NTCIP.EssPrecipRate                    OPTIONAL,
   item8-8        NTCIP.EssPrecipSituation               OPTIONAL,
   item8-9        NTCIP.EssPrecipYesNo                   OPTIONAL,
   item8-10       NTCIP.EssSolarRadiation                OPTIONAL,
   item8-11       ITIS.ITIScodes                         OPTIONAL,
   ...
   }


-- DF_AntennaOffsetSet (Desc Name) Record 4
AntennaOffsetSet ::= OCTET STRING (SIZE(4))
   -- defined as:
   -- SEQUENCE {
   -- antOffsetX  INTEGER (-8191..8191),
                  -- 14 bits in length
                  -- units of 1cm from center
                  -- 8191 to be used for unavailable
   -- antOffsetY  INTEGER (-255..255),
                  -- 9 bits in length
                  -- units of 1cm from center
                  -- 255 to be used for unavailable
   -- antOffsetZ  INTEGER (0..511)
                  -- 9 bits in length
                  -- units of 1cm from ground
                  -- 511 to be used for unavailable
   -- }


-- DF_Approach (Desc Name) Record 5
Approach ::=  SEQUENCE {
   name           DescriptiveName OPTIONAL,
   id             ApproachNumber  OPTIONAL,
   drivingLanes   SEQUENCE (SIZE(0..32)) OF
                  VehicleReferenceLane OPTIONAL,
   computedLanes  SEQUENCE (SIZE(0..32)) OF
                  VehicleComputedLane OPTIONAL,
   trainsAndBuses SEQUENCE (SIZE(0..32)) OF
                  SpecialLane OPTIONAL,
   barriers       SEQUENCE (SIZE(0..32)) OF
                  BarrierLane OPTIONAL,
   crosswalks     SEQUENCE (SIZE(0..32)) OF
                  CrosswalkLane OPTIONAL,
   ...
   }


-- DF_ApproachesObject (Desc Name) Record 6
ApproachObject ::=  SEQUENCE {
   refPoint     Position3D OPTIONAL,
                -- optional reference from which subsequent
                -- data points in this link are offset
   laneWidth    LaneWidth  OPTIONAL,
                -- reference width used by subsequent
                -- lanes until a new width is given
   approach     Approach OPTIONAL,
                -- list of Approaches and their lanes
```

```
   egress       Approach OPTIONAL,
                -- list of Egresses and thier lanes
   ...
   }


-- DF_BarrierLane (Desc Name) Record 7
BarrierLane ::=  SEQUENCE {
   laneNumber          LaneNumber,
   laneWidth           LaneWidth  OPTIONAL,
   barrierAttributes   BarrierAttributes,
   nodeList            NodeList,
                       -- path details of the Barrier
   ...
   }


-- DF_BrakeSystemStatus (Desc Name) Record 8
BrakeSystemStatus ::= OCTET STRING (SIZE(2))
   -- Encoded with the packed content of:
   -- SEQUENCE {
   --   wheelBrakes        BrakeAppliedStatus,
   --                      -x- 4 bits
   --   wheelBrakesUnavailable  BOOL
   --                      -x- 1 bit (1=true)
   --   spareBit
   --                      -x- 1 bit, set to zero
   --   traction           TractionControlState,
   --                      -x- 2 bits
   --   abs                AntiLockBrakeStatus,
   --                      -x- 2 bits
   --   scs                StabilityControlStatus,
   --                      -x- 2 bits
   --   brakeBoost         BrakeBoostApplied,
   --                      -x- 2 bits
   --   auxBrakes          AuxiliaryBrakeStatus,
   --                      -x- 2 bits
   --   }


-- DF_BSM_Blob (Desc Name) Record 9
BSMblob ::= OCTET STRING (SIZE(38))
   -- made up of the following 38 packed bytes:
   -- msgCnt      MsgCount,             -x- 1 byte
   -- id          TemporaryID,          -x- 4 bytes
   -- secMark     DSecond,              -x- 2 bytes

   -- lat         Latitude,             -x- 4 bytes
   -- long        Longitude,            -x- 4 bytes
   -- elev        Elevation,            -x- 2 bytes
   -- accuracy    PositionalAccuracy,   -x- 4 bytes

   -- speed       TransmissionAndSpeed, -x- 2 bytes
   -- heading     Heading,              -x- 2 byte
   -- angle       SteeringWheelAngle    -x- 1 byte
   -- accelSet    AccelerationSet4Way,  -x- accel set (four way) 7 bytes

   -- brakes      BrakeSystemStatus,    -x- 2 bytes
   -- size        VehicleSize,          -x- 3 bytes


-- DF_BumperHeights (Desc Name) Record 10
BumperHeights ::=  SEQUENCE {
   frnt       BumperHeightFront,
   rear       BumperHeightRear
   }


-- DF_Circle (Desc Name) Record 11
Circle ::=  SEQUENCE {
   center   Position3D,
   raduis   CHOICE {
               radiusSteps  INTEGER (0..32767),
                              -- in unsigned values where
                              -- the LSB is in units of 2.5 cm
               miles        INTEGER (1..2000),
               km           INTEGER (1..5000)
            } --# UNTAGGED
```

```
   }


-- DF_ConfidenceSet (Desc Name) Record 12
ConfidenceSet ::=  SEQUENCE {
   accelConfidence      AccelSteerYawRateConfidence OPTIONAL,
   speedConfidence      SpeedandHeadingandThrottleConfidence OPTIONAL,
   timeConfidence       TimeConfidence OPTIONAL,
   posConfidence        PositionConfidenceSet OPTIONAL,
   steerConfidence      SteeringWheelAngleConfidence OPTIONAL,
   throttleConfidence   ThrottleConfidence OPTIONAL,
   ... -- # LOCAL_CONTENT
   }


-- DF_ConnectsTo (Desc Name) Record 13
ConnectsTo ::= OCTET STRING (SIZE(2..32))
   -- sets of 2 byte pairs,
   -- the first byte is a lane number
   -- the second byte is a LaneManeuverCode


-- DF_CrosswalkLane (Desc Name) Record 14
CrosswalkLane ::=  SEQUENCE {
   laneNumber       LaneNumber,
   laneWidth        LaneWidth  OPTIONAL,
   laneAttributes   CrosswalkLaneAttributes,
   nodeList         NodeList,
                    -- path details of the lane
                    -- note that this may cross or pass
                    -- by driven lanes
   keepOutList      NodeList OPTIONAL,
                    -- no stop points along the path
                    -- typically the end points unless
                    -- islands are represented in the path
   connectsTo       ConnectsTo  OPTIONAL,
                    -- a list of other lanes and their
                    -- turning use by this lane
   ...
   }


-- DF_DataParameters (Desc Name) Record 15
DataParameters ::=  SEQUENCE {
   processMethod     IA5String(SIZE(1..255)) OPTIONAL,
   processAgency     IA5String(SIZE(1..255)) OPTIONAL,
   lastCheckedDate   IA5String(SIZE(1..255)) OPTIONAL,
   geiodUsed         IA5String(SIZE(1..255)) OPTIONAL,
   ... -- # LOCAL_CONTENT
   }


-- DF_DDate (Desc Name) Record 16
DDate ::=  SEQUENCE {
   year    DYear,            -- 2 bytes
   month   DMonth,           -- 1 byte
   day     DDay              -- 1 byte
   }


-- DF_DDateTime (Desc Name) Record 17
DDateTime ::=  SEQUENCE {
   year    DYear    OPTIONAL,    -- 2 bytes
   month   DMonth   OPTIONAL,    -- 1 byte
   day     DDay     OPTIONAL,    -- 1 byte
   hour    DHour    OPTIONAL,    -- 1 byte
   minute  DMinute  OPTIONAL,    -- 1 byte
   second  DSecond  OPTIONAL     -- 2 bytes
   }


-- DF_DFullTime (Desc Name) Record 18
DFullTime ::=  SEQUENCE {
   year    DYear,            -- 2 bytes
   month   DMonth,           -- 1 byte
   day     DDay,             -- 1 byte
```

```
   hour    DHour,            -- 1 byte
   minute  DMinute           -- 1 byte
   }


-- DF_DMonthDay (Desc Name) Record 19
DMonthDay ::=  SEQUENCE {
   month   DMonth,           -- 1 byte
   day     DDay              -- 1 byte
   }


-- DF_DTime (Desc Name) Record 20
DTime ::=  SEQUENCE {
   hour    DHour,            -- 1 byte
   minute  DMinute,          -- 1 byte
   second  DSecond           -- 2 bytes
   }


-- DF_DYearMonth (Desc Name) Record 21
DYearMonth ::=  SEQUENCE {
   year    DYear,            -- 2 bytes
   month   DMonth            -- 1 byte
   }


-- DF_ITIS_Phrase_ExitService (Desc Name) Record 22
ExitService ::=  SEQUENCE (SIZE(1..10)) OF SEQUENCE {
  item CHOICE    {
       itis   ITIS.ITIScodes,
       text   IA5String  (SIZE(1..16))
       } -- # UNTAGGED
  }


-- DF_FullPositionVector (Desc Name) Record 23
FullPositionVector ::=  SEQUENCE {
   utcTime             DDateTime OPTIONAL,   -- time with mSec precision
   long                Longitude,            -- 1/10th microdegree
   lat                 Latitude,             -- 1/10th microdegree
   elevation           Elevation OPTIONAL,   -- 3 bytes, 0.1 m
   heading             Heading OPTIONAL,
   speed               TransmissionAndSpeed OPTIONAL,
   posAccuracy         PositionalAccuracy OPTIONAL,
   timeConfidence      TimeConfidence OPTIONAL,
   posConfidence       PositionConfidenceSet OPTIONAL,
   speedConfidence     SpeedandHeadingandThrottleConfidence OPTIONAL,
   ... -- # LOCAL_CONTENT
   }


-- DF_ITIS_Phrase_GenericSignage (Desc Name) Record 24
GenericSignage ::=  SEQUENCE (SIZE(1..10)) OF SEQUENCE {
  item CHOICE    {
       itis   ITIS.ITIScodes,
       text   IA5String  (SIZE(1..16))
       } -- # UNTAGGED
  }


-- DF_Intersection (Desc Name) Record 25
Intersection ::=  SEQUENCE {
   name        DescriptiveName OPTIONAL,
   id          IntersectionID,
                                -- a gloablly unique value,
                                -- the upper bytes of which may not
                                -- be sent if the context is known
   refPoint    Position3D OPTIONAL,
                                -- the reference from which subsequent
                                -- data points are offset until a new
                                -- point is used.
   refInterNum IntersectionID OPTIONAL,
                                -- present only if this is a computed
                                -- intersection instance
   orientation Heading OPTIONAL,
```

```
                                -- present only if this is a computed
                                -- intersection instance

   laneWidth   LaneWidth  OPTIONAL,
                                -- reference width used by subsequent
                                -- lanes until a new width is given
   type        IntersectionStatusObject OPTIONAL,
                                -- data about the intersection type
   approaches SEQUENCE (SIZE(1..32)) OF
               ApproachObject,
                                -- data about one or more approaches
                                -- (lane data is found here)
   preemptZones SEQUENCE (SIZE(1..32)) OF
               SignalControlZone OPTIONAL,
                                -- data about one or more
                                -- preempt zones
   priorityZones SEQUENCE (SIZE(1..32)) OF
               SignalControlZone OPTIONAL,
                                -- data about one or more
                                -- priority zones
   ...
   }


-- DF_IntersectionState (Desc Name) Record 26
IntersectionState ::=  SEQUENCE {
   name        DescriptiveName OPTIONAL,
               -- human readable name for intersection
               -- to be used only in debug mode
   id          IntersectionID,
               -- this provided a unique mapping to the
               -- intersection map in question
               -- which provides complete location
               -- and approach/move/lane data
   status      IntersectionStatusObject,
               -- general status of the controller
   timeStamp   TimeMark OPTIONAL,
               -- the point in local time that
               -- this message was constructed
   lanesCnt    INTEGER(1..255) OPTIONAL,
               -- number of states to follow (not always
               -- one per lane because sign states may be shared)
   states      SEQUENCE (SIZE(1..255)) OF MovementState,
               -- each active Movement/lane is given in turn
               -- and contains its state, and seconds
               -- to the next event etc.
   priority    SignalState OPTIONAL,
               -- the active priority state data, if present
   preempt     SignalState OPTIONAL,
               -- the active preemption state data, if present

   ... -- # LOCAL_CONTENT
   }


-- DF_J1939-Data Items (Desc Name) Record 27
J1939data ::=  SEQUENCE {
   -- Tire conditions
   tires SEQUENCE (SIZE(0..16)) OF SEQUENCE {
      location              TireLocation              OPTIONAL,
      pressure              TirePressure              OPTIONAL,
      temp                  TireTemp                  OPTIONAL,
      wheelSensorStatus     WheelSensorStatus         OPTIONAL,
      wheelEndElectFault    WheelEndElectFault        OPTIONAL,
      leakageRate           TireLeakageRate           OPTIONAL,
      detection             TirePressureThresholdDetection OPTIONAL,
      ...
      } OPTIONAL,
   -- Vehicle Weight by axle
   axle SEQUENCE (SIZE(0..16)) OF SEQUENCE {
      location              AxleLocation              OPTIONAL,
      weight                AxleWeight                OPTIONAL,
      ...
      } OPTIONAL,
   trailerWeight            TrailerWeight             OPTIONAL,
   cargoWeight              CargoWeight               OPTIONAL,
   steeringAxleTemperature  SteeringAxleTemperature   OPTIONAL,
   driveAxleLocation        DriveAxleLocation         OPTIONAL,
   driveAxleLiftAirPressure DriveAxleLiftAirPressure  OPTIONAL,
   driveAxleTemperature     DriveAxleTemperature      OPTIONAL,
   driveAxleLubePressure    DriveAxleLubePressure     OPTIONAL,
```

```
   steeringAxleLubePressure SteeringAxleLubePressure  OPTIONAL,
   ...
   }


-- DF_MovementState (Desc Name) Record 28
MovementState ::=  SEQUENCE {
   -- The MovementNumber is contained in the enclosing DF.
   movementName     DescriptiveName OPTIONAL,
                    -- uniquely defines movement by name
   laneCnt          LaneCount OPTIONAL,
                    -- the number of lanes to follow
   laneSet          LaneSet,
                    -- each encoded as a LaneNumber,
                    -- the collection of lanes, by num,
                    -- to which this state data applies
  -- For the current movement State, you may CHOICE one of the below:
          currState     SignalLightState OPTIONAL,
                        -- the state of a Motorised lane
          pedState      PedestrianSignalState OPTIONAL,
                        -- the state of a Pedestrian type lane
          specialState  SpecialSignalState OPTIONAL,
                        -- the state of a special type lane
                        -- such as a dedicated train lane

   timeToChange     TimeMark,
                    -- the point in time this state will change
   stateConfidence  StateConfidence  OPTIONAL,

   -- Yellow phase time intervals
   -- (used for motorised vehicle lanes and pedestrian lanes)
   -- For the yellow Signal State, a CHOICE of one of the below:
           yellState     SignalLightState OPTIONAL,
                         -- the next state of a
                         -- Motorised lane
           yellPedState  PedestrianSignalState OPTIONAL,
                         -- the next state of a
                         -- Pedestrian type lane

   yellTimeToChange     TimeMark  OPTIONAL,
   yellStateConfidence  StateConfidence  OPTIONAL,

   -- below items are all optional based on use and context
   -- some are used only for ped lane types
   vehicleCount  ObjectCount OPTIONAL,
   pedDetect     PedestrianDetect  OPTIONAL,
                 -- true if ANY ped are detected crossing
                 -- the above lanes
   pedCount      ObjectCount OPTIONAL,
                 -- est count of peds
   ... -- # LOCAL_CONTENT
   }


-- DF_NodeList (Desc Name) Record 29
NodeList ::=  SEQUENCE (SIZE(1..64)) OF Offsets
   -- the Position3D ref point (starting point or anchor)
   -- is found in the outer object.
   -- Offsets are additive from the last point.


-- DF_Offsets (Desc Name) Record 30
Offsets ::= OCTET STRING (SIZE(4..8))
   -- Made up of
   -- SEQUENCE {
   -- xOffset  INTEGER (-32767..32767),
   -- yOffset  INTEGER (-32767..32767),
   -- if 6 or 8 bytes in length:
   -- zOffset  INTEGER (-32767..32767) OPTIONAL,
            -- all above in signed values where
            -- the LSB is in units of 1.0 cm

   -- if 8 bytes in length:
   -- width    LaneWidth              OPTIONAL
   -- a length of 7 bytes is never used
   -- }


-- DF_PathHistory (Desc Name) Record 31
```

```
PathHistory ::=  SEQUENCE {
   initialPosition  FullPositionVector   OPTIONAL,
   currGPSstatus    GPSstatus            OPTIONAL,
   itemCnt          Count                OPTIONAL,
                    -- Limited to range 1 to 23
                    -- number of points in set to follow
   crumbData  CHOICE {
              -- select one of the possible data sets to be used

              pathHistoryPointSets-01 SEQUENCE (SIZE(1..23)) OF
                    PathHistoryPointType-01,
                    -- made up of sets of the: PathHistoryPointType-1
                    -- a set of all data elements, it is
                    -- non-uniform in size, each item tagged in BER

              pathHistoryPointSets-02 OCTET STRING (SIZE(15..345)),
                    -- made up of sets of the: PathHistoryPointType-02
                    -- sets of all data elements including:
                    -- lat, long, elev, time, accuracy, heading, and speed
                    -- offsets sent as a packed blob of 15 bytes per point

              pathHistoryPointSets-03 OCTET STRING (SIZE(12..276)),
                    -- made up of sets of the: PathHistoryPointType-03
                    -- sets of the following data elements:
                    -- lat, long, elev, time, and accuracy
                    -- offsets sent as a packed blob of 12 bytes per point

              pathHistoryPointSets-04 OCTET STRING (SIZE(8..184)),
                    -- made up of sets of the: PathHistoryPointType-04
                    -- sets of the following data elements:
                    -- lat, long, elev, and time
                    -- offsets sent as a packed blob of 8 bytes per point

              pathHistoryPointSets-05 OCTET STRING (SIZE(10..230)),
                    -- made up of sets of the: PathHistoryPointType-05
                    -- sets of the following data elements:
                    -- lat, long, elev, and accuracy
                    -- offsets sent as a packed blob of 10 bytes per point

              pathHistoryPointSets-06 OCTET STRING (SIZE(6..138)),
                    -- made up of sets of the: PathHistoryPointType-06
                    -- sets of the following data elements:
                    -- lat, long, and elev
                    -- offsets sent as a packed blob of 6 bytes per point

              pathHistoryPointSets-07 OCTET STRING (SIZE(11..242)),
                    -- made up of sets of the: PathHistoryPointType-07
                    -- sets of the following data elements:
                    -- lat, long, time, and accuracy
                    -- offsets sent as a packed blob of 10.5 bytes per point

              pathHistoryPointSets-08 OCTET STRING (SIZE(7..161)),
                    -- made up of sets of the: PathHistoryPointType-08
                    -- sets of the following data elements:
                    -- lat, long, and time
                    -- offsets sent as a packed blob of 7 bytes per point

              pathHistoryPointSets-09 OCTET STRING (SIZE(9..196)),
                    -- made up of sets of the: PathHistoryPointType-09
                    -- sets of the following data elements:
                    -- lat, long, and accuracy
                    -- offsets sent as a packed blob of 8.5 bytes per point

              pathHistoryPointSets-10 OCTET STRING (SIZE(5..104))
                    -- made up of sets of the: PathHistoryPointType-10
                    -- sets of the following data elements:
                    -- lat and long
                    -- offsets sent as a packed blob of 4.5 bytes per point

              },
   ... -- # LOCAL_CONTENT
   }


-- DF_PathHistoryPointType-01 (Desc Name) Record 32
PathHistoryPointType-01 ::=  SEQUENCE {
   latOffset  INTEGER (-131072..131071),
               -- in 1/10th micro degrees
               -- value  131071 to be used for  131071 or greater
               -- value -131071 to be used for -131071 or less
               -- value -131072 to be used for unavailable lat or long
```

```
   longOffset INTEGER (-131072..131071),
              -- in 1/10th micro degrees
              -- value  131071 to be used for  131071 or greater
              -- value -131071 to be used for -131071 or less
              -- value -131072 to be used for unavailable lat or long

    elevationOffset INTEGER (-2048..2047) OPTIONAL,
              -- LSB units of of 10 cm
              -- value  2047 to be used for  2047 or greater
              -- value -2047 to be used for -2047 or greater
              -- value -2048 to be unavailable

    timeOffset INTEGER (1..65535) OPTIONAL,
              -- LSB units of of 10 mSec
              -- value  65534 to be used for 65534 or greater
              -- value  65535 to be unavailable

    posAccuracy PositionalAccuracy OPTIONAL,
              -- four packed bytes

    heading    INTEGER (-128..127) OPTIONAL,
              -- where the LSB is in
              -- units of 1.5 degrees
              -- value -128 for unavailable
              -- not an offset value

    speed      TransmissionAndSpeed  OPTIONAL
              -- upper bits encode transmission
              -- where the LSB is in
              -- units of 0.02 m/s
              -- not an offset value
   }


-- DF_PathHistoryPointType-02 (Desc Name) Record 33
PathHistoryPointType-02 ::= OCTET STRING (SIZE(15))
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- elevationOffset    INTEGER (-2048..2047), (12 signed bits)
    -- LSB units of 10 cm
    -- value  2047 to be used for  2047 or greater
    -- value -2047 to be used for -2047 or greater
    -- value -2048 to be unavailable

   -- timeOffset INTEGER (0..65535), (16 unsigned bits)
    -- LSB units of of 10 mSec
    -- value  65534 to be used for 65534 or greater
    -- value  65535 to be unavailable

   -- accuracy   PositionalAccuracy
    -- four packed bytes

   -- heading    INTEGER (-128..127), (8 signed bits)
    -- where the LSB is in
    -- units of 1.5 degrees
    -- value -128 for unavailable
    -- not an offset value

   -- speed      TransmissionAndSpeed (16 encoded bits)
    -- upper bits encode transmission
    -- where the LSB is in
    -- units of 0.02 m/s
    -- not an offset value


-- DF_PathHistoryPointType-03 (Desc Name) Record 34
PathHistoryPointType-03 ::= OCTET STRING (SIZE(12))
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long
```

```
   -- elevationOffset    INTEGER (-2048..2047), (12 signed bits)
    -- LSB units of 10 cm
    -- value  2047 to be used for  2047 or greater
    -- value -2047 to be used for -2047 or greater
    -- value -2048 to be unavailable

   -- timeOffset INTEGER (0..65535), (16 unsigned bits)
    -- LSB units of of 10 mSec
    -- value  65534 to be used for 65534 or greater
    -- value  65535 to be unavailable

   -- accuracy   PositionalAccuracy
    -- four packed bytes


-- DF_PathHistoryPointType-04 (Desc Name) Record 35
PathHistoryPointType-04 ::= OCTET STRING (SIZE(8))
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- elevationOffset    INTEGER (-2048..2047), (12 signed bits)
    -- LSB units of 10 cm
    -- value  2047 to be used for  2047 or greater
    -- value -2047 to be used for -2047 or greater
    -- value -2048 to be unavailable

   -- timeOffset INTEGER (0..65535), (16 unsigned bits)
    -- LSB units of of 10 mSec
    -- value  65534 to be used for 65534 or greater
    -- value  65535 to be unavailable


-- DF_PathHistoryPointType-05 (Desc Name) Record 36
PathHistoryPointType-05 ::= OCTET STRING (SIZE(10))
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- elevationOffset    INTEGER (-2048..2047), (12 signed bits)
    -- LSB units of 10 cm
    -- value  2047 to be used for  2047 or greater
    -- value -2047 to be used for -2047 or greater
    -- value -2048 to be unavailable

   -- accuracy   PositionalAccuracy
    -- four packed bytes


-- DF_PathHistoryPointType-06 (Desc Name) Record 37
PathHistoryPointType-06 ::= OCTET STRING (SIZE(6))
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- elevationOffset    INTEGER (-2048..2047), (12 signed bits)
    -- LSB units of 10 cm
    -- value  2047 to be used for  2047 or greater
    -- value -2047 to be used for -2047 or greater
    -- value -2048 to be unavailable


-- DF_PathHistoryPointType-07 (Desc Name) Record 38
PathHistoryPointType-07 ::= OCTET STRING (SIZE(11)) -- in fact 10.5
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
```

```
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- timeOffset INTEGER (0..65535), (16 unsigned bits)
    -- LSB units of of 10 mSec
    -- value  65534 to be used for 65534 or greater
    -- value  65535 to be unavailable

   -- accuracy   PositionalAccuracy
    -- four packed bytes


-- DF_PathHistoryPointType-08 (Desc Name) Record 39
PathHistoryPointType-08 ::= OCTET STRING (SIZE(7)) -- in fact 6.5
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- timeOffset INTEGER (0..65535), (16 unsigned bits)
    -- LSB units of of 10 mSec
    -- value  65534 to be used for 65534 or greater
    -- value  65535 to be unavailable


-- DF_PathHistoryPointType-09 (Desc Name) Record 40
PathHistoryPointType-09 ::= OCTET STRING (SIZE(9)) -- in fact 8.5
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long

   -- accuracy   PositionalAccuracy
    -- four packed bytes


-- DF_PathHistoryPointType-10 (Desc Name) Record 41
PathHistoryPointType-10 ::= OCTET STRING (SIZE(5))   -- in fact 4.5
   -- To be made up of packed bytes as follows:
   -- latOffset  INTEGER (-131072..131071) (18 signed bits)
   -- longOffset INTEGER (-131072..131071) (18 signed bits)
    -- in 1/10th micro degrees
    -- value  131071 to be used for  131071 or greater
    -- value -131071 to be used for -131071 or less
    -- value -131072 to be used for unavailable lat or long


-- DF_PathPrediction (Desc Name) Record 42
PathPrediction ::=  SEQUENCE {
   radiusOfCurve INTEGER (-32767..32767),
                 -- LSB units of 10cm
                 -- straight path to use value of 32767
   confidence    INTEGER (0..200),
                 -- LSB units of 0.5 percent

   ... -- # LOCAL_CONTENT
   }


-- DF_Position3D (Desc Name) Record 43
Position3D ::=  SEQUENCE {
   lat         Latitude,   -- in 1/10th micro degrees
   long        Longitude,  -- in 1/10th micro degrees
   elevation   Elevation   OPTIONAL
   }


-- DF_PositionalAccuracy (Desc Name) Record 44
PositionalAccuracy ::= OCTET STRING (SIZE(4))
```

```
  -- And the bytes defined as folllows

  -- Byte 1: semi-major accuracy at one standard dev
  -- range 0-12.7 meter, LSB = .05m
  -- 0xFE=254=any value equal or greater than 12.70 meter
  -- 0xFF=255=unavailable semi-major value

  -- Byte 2: semi-minor accuracy at one standard dev
  -- range 0-12.7 meter, LSB = .05m
  -- 0xFE=254=any value equal or greater than 12.70 meter
  -- 0xFF=255=unavailable semi-minor value

  -- Bytes 3-4: orientation of semi-major axis
  -- relative to true north (0~359.9945078786 degrees)
  -- LSB units of 360/65535 deg  = 0.0054932479
  -- a value of 0x0000 =0 shall be 0 degrees
  -- a value of 0x0001 =1 shall be 0.0054932479degrees
  -- a value of 0xFFFE =65534 shall be 359.9945078786 deg
  -- a value of 0xFFFF =65535 shall be used for orientation unavailable
  -- (In NMEA GPGST)



-- DF_PositionConfidenceSet (Desc Name) Record 45
PositionConfidenceSet ::= OCTET STRING (SIZE(1))
-- To be encoded as:
-- SEQUENCE {
--   pos        PositionConfidence,
--              -x- 4 bits, for both horizontal directions
--   elevation  ElevationConfidence
--              -x- 4 bits
--   }



-- DF_RegionList (Desc Name) Record 46
RegionList ::=  SEQUENCE (SIZE(1..64)) OF RegionOffsets
   -- the Position3D ref point (starting point or anchor)
   -- is found in the outer object.



-- DF_RegionOffsets (Desc Name) Record 47
RegionOffsets ::=  SEQUENCE {
   xOffset  INTEGER (-32767..32767),
   yOffset  INTEGER (-32767..32767),
   zOffset  INTEGER (-32767..32767) OPTIONAL
            -- all in signed values where
            -- the LSB is in units of 1 meter
   }



-- DF_RegionPointSet (Desc Name) Record 48
RegionPointSet ::=  SEQUENCE {
   anchor           Position3D  OPTIONAL,
   nodeList         RegionList,
                    -- path details of the regions outline
   ...
   }



-- DF_RoadSignID (Desc Name) Record 49
RoadSignID ::=  SEQUENCE {
   position        Position3D,
                   -- Location of sign
   viewAngle       HeadingSlice,
                   -- Vehicle direction of travel while
                   -- facing active side of sign
   mutcdCode       MUTCDCode OPTIONAL,
                   -- Tag for MUTCD code or "generic sign"
   crc             MsgCRC OPTIONAL
                   -- Used to provide a check sum
   }



-- DF_RTCMHeader (Desc Name) Record 50
RTCMHeader ::= OCTET STRING (SIZE(5))
   -- defined as:
   -- SEQUENCE {
   -- status     GPSstatus,
```

```
                -- to occupy 1 byte
    -- offsetSet  AntennaOffsetSet
                -- to occupy 4 bytes
    -- }


-- DF_RTCMmsg (Desc Name) Record 51
RTCMmsg ::=  SEQUENCE {
    rev        RTCM-Revision OPTIONAL,
    rtcmID     RTCM-ID       OPTIONAL,
               -- the message and sub-message type, as
               -- defined in the RTCM revision being used
    payload    RTCM-Payload,
               -- the payload bytes
    ... -- # LOCAL_CONTENT
    }


-- DF_RTCMPackage (Desc Name) Record 52
RTCMPackage ::=  SEQUENCE {
    anchorPoint FullPositionVector OPTIONAL,
    -- precise observer position, if needed

    rtcmHeader  RTCMHeader,
    -- an octet blob consisting of:
    -- one byte with:
    -- GPSstatus
    -- 4 bytes with:
    -- AntennaOffsetSet  containing x,y,z data

    -- note that a max of 16 satellites are allowed
    msg1001    OCTET STRING (SIZE(16..124))  OPTIONAL,
               -- pRange data GPS L1
    msg1002    OCTET STRING (SIZE(18..156))  OPTIONAL,
               -- pRange data GPS L1

    msg1003    OCTET STRING (SIZE(21..210))  OPTIONAL,
                -- pRange data GPS L1, L2
    msg1004    OCTET STRING (SIZE(24..258))  OPTIONAL,
                -- pRange data GPS L1, L2

    msg1005    OCTET STRING (SIZE(19))  OPTIONAL,
               -- observer station data
    msg1006    OCTET STRING (SIZE(21))  OPTIONAL,
               -- observer station data

    msg1007    OCTET STRING (SIZE(5..36))  OPTIONAL,
               -- antenna of observer station data
    msg1008    OCTET STRING (SIZE(6..68))  OPTIONAL,
               -- antenna of observer station data

    msg1009    OCTET STRING (SIZE(16..136))  OPTIONAL,
                -- pRange data GLONASS L1
    msg1010    OCTET STRING (SIZE(18..166))  OPTIONAL,
                -- pRange data GLONASS L1

    msg1011    OCTET STRING (SIZE(21..222))  OPTIONAL,
                -- pRange data GLONASS L1, L2
    msg1012    OCTET STRING (SIZE(24..268))  OPTIONAL,
                -- pRange data GLONASS L1, L2

    msg1013    OCTET STRING (SIZE(13..27))  OPTIONAL,
                -- system parameters data

    ..., -- # LOCAL_CONTENT
    -- The below items shall never be sent
    -- over WSM stack encoding (other encodings may be used)
    -- and may be removed from the ASN

    msg1014    OCTET STRING (SIZE(15))  OPTIONAL,
               -- Network Aux Station (NAS) data
    msg1015    OCTET STRING (SIZE(13..69))  OPTIONAL,
               -- Ionospheric Correction data
    msg1016    OCTET STRING (SIZE(14..81))  OPTIONAL,
               -- Geometry Correction data
    msg1017    OCTET STRING (SIZE(16..115))  OPTIONAL,
               -- Combined Ionospheric and Geometry data

    -- msg1018 is reserved at this time

    msg1019    OCTET STRING (SIZE(62))  OPTIONAL,
```

```
            -- Satellite Ephermeris data
   msg1020  OCTET STRING (SIZE(45))  OPTIONAL,
            -- Satellite Ephermeris data
   msg1021  OCTET STRING (SIZE(62))  OPTIONAL,
            -- Helmert-Abridged Molodenski Transform data
   msg1022  OCTET STRING (SIZE(75))  OPTIONAL,
            -- Molodenski-Badekas Transform data
   msg1023  OCTET STRING (SIZE(73))  OPTIONAL,
            -- Ellipse Residuals data
   msg1024  OCTET STRING (SIZE(74))  OPTIONAL,
            -- Plane-Grid Residuals data
   msg1025  OCTET STRING (SIZE(25))  OPTIONAL,
            -- Non-Lab Conic Project data
   msg1026  OCTET STRING (SIZE(30))  OPTIONAL,
            -- Lab Conic Conform Project data
   msg1027  OCTET STRING (SIZE(33))  OPTIONAL,
            -- Ob Mercator Project data

   -- msg1028 is reserved at this time

   msg1029  OCTET STRING (SIZE(10..69))  OPTIONAL,
            -- Unicode test type data
   msg1030  OCTET STRING (SIZE(14..105))  OPTIONAL,
            -- GPS Residuals data
   msg1031  OCTET STRING (SIZE(15..107))  OPTIONAL,
            -- GLONASS Residuals data
   msg1032  OCTET STRING (SIZE(20))  OPTIONAL,
            -- Ref Station Position data

   -- Proprietary Data content (msg40xx to msg4095)
   -- may be added as needed

   ... -- # LOCAL_CONTENT
   }


-- DF_Sample (Desc Name) Record 53
Sample ::=  SEQUENCE {
   sampleStart  INTEGER(0..255),   --  Sample Starting Point
   sampleEnd    INTEGER(0..255)    --  Sample Ending Point
   }


-- DF_ShapePointSet (Desc Name) Record 54
ShapePointSet ::=  SEQUENCE {
   anchor          Position3D     OPTIONAL,
   laneWidth       LaneWidth      OPTIONAL,
   directionality  DirectionOfUse OPTIONAL,
   nodeList        NodeList,      -- path details of the lane and width
   ...
   }


-- DF_SignalControlZone (Desc Name) Record 55
SignalControlZone ::=  SEQUENCE {
   name           DescriptiveName OPTIONAL,
                  -- used only for debugging
   pValue         SignalReqScheme,
                  -- preempt or priorty value (0..7),
                  -- and any strategy value to be used

   data           CHOICE {

         laneSet  SEQUENCE (SIZE(1..32)) OF LaneNumber,
                  -- a seq of of defined LaneNumbers,
                  -- to be used with this p value
                  -- see thier nodelists for paths

         zones    SEQUENCE (SIZE(1..32)) OF SEQUENCE {

             enclosed  SEQUENCE (SIZE(1..32)) OF LaneNumber OPTIONAL,
                             -- lanes in this region
             laneWidth       LaneWidth  OPTIONAL,
             nodeList        NodeList,
                             -- path details of
                             -- the region starting from
                             -- the stop line
             ...
             }
             -- Note: unlike a nodelist for lanes,
```

```
            -- zones may overlap by a considerable degree
        },
    ... -- # LOCAL_CONTENT
    }


-- DF_SignalRequest (Desc Name) Record 56
SignalRequest ::=  SEQUENCE {
    -- the regionally unique ID of the target intersection
    id     IntersectionID,  --  intersection ID

    -- Below present only when canceling a prior request
    isCancel   SignalReqScheme OPTIONAL,

    -- In typical use either a SignalReqScheme
    -- or a lane number would be given, this
    -- indicates the scheme to use or the
    -- path through the intersection
    -- to the degree it is known.
    -- Note that SignalReqScheme can hold either
    -- a preempt or a priority value.
    requestedAction  SignalReqScheme OPTIONAL,
                            --  preempt ID or the
                            --  priority ID
                            --  (and strategy)
    inLane      LaneNumber OPTIONAL,
                            --  approach Lane
    outLane     LaneNumber OPTIONAL,
                            --  egress Lane
    type        NTCIPVehicleclass,
                            --  Two 4 bit nibbles as:
                            --  NTCIP vehicle class type
                            --  NTCIP vehicle class level

    -- any validation string used by the system
    codeWord    CodeWord OPTIONAL,
    ...
    }


-- DF_Snapshot (Desc Name) Record 57
Snapshot ::=  SEQUENCE {
    thePosition  FullPositionVector,
                 -- data of the position and speed,
    safetyExt    VehicleSafetyExtension OPTIONAL,
    datSet       VehicleStatus          OPTIONAL,
                 -- a seq of data frames
                 -- which encodes the data
  ... -- # LOCAL_CONTENT
    }


-- DF_SnapshotDistance (Desc Name) Record 58
SnapshotDistance ::=  SEQUENCE {
    d1   INTEGER(0..999),    -- meters
    s1   INTEGER(0..50),     -- meters\second
    d2   INTEGER(0..999),    -- meters
    s2   INTEGER(0..50)      -- meters\second
    }


-- DF_SnapshotTime (Desc Name) Record 59
SnapshotTime ::=  SEQUENCE {
    t1   INTEGER(1..99),
         -- m/sec - the instantaneous speed when the
         -- calculation is performed
    s1   INTEGER(0..50),
         -- seconds
    t2   INTEGER(1..99),
         -- m/sec - the instantaneous speed when the
         -- calculation is performed
    s2   INTEGER(0..50)
         -- seconds
    }


-- DF_SpecialLane (Desc Name) Record 60
SpecialLane ::=  SEQUENCE {
```

```
   laneNumber       LaneNumber,
   laneWidth        LaneWidth  OPTIONAL,
   laneAttributes   SpecialLaneAttributes,
   nodeList         NodeList,
                    -- path details of the lane and width
   keepOutList      NodeList OPTIONAL,
                    -- no stop points along the path
   connectsTo       ConnectsTo  OPTIONAL,
                    -- a list of other lanes and their
                    -- turning use by this lane
   ...
   }


-- DF_Speed_Heading_Throttle_Confidence (Desc Name) Record 61
SpeedandHeadingandThrottleConfidence ::= OCTET STRING (SIZE(1))
-- to be packed as follows:
-- SEQUENCE {
--   heading   HeadingConfidence,   -x- 3 bits
--   speed     SpeedConfidence,     -x- 3 bits
--   throttle  ThrottleConfidence   -x- 2 bits
--   }


-- DF_ITIS_Phrase_SpeedLimit (Desc Name) Record 62
SpeedLimit ::=  SEQUENCE (SIZE(1..10)) OF SEQUENCE {
  item CHOICE    {
       itis   ITIS.ITIScodes,
       text   IA5String  (SIZE(1..16))
       } -- # UNTAGGED
  }


-- DF_TransmissionAndSpeed (Desc Name) Record 63
TransmissionAndSpeed ::= OCTET STRING (SIZE(2))
    -- Bits 14~16 to be made up of the data element
    -- DE_TransmissionState
    -- Bits 1~13 to be made up of the data element
    -- DE_Speed


-- DF_ValidRegion (Desc Name) Record 64
ValidRegion ::=  SEQUENCE {

   direction         HeadingSlice,
                     -- field of view over which this applies,
   extent            Extent OPTIONAL,
                     -- the spatial distance over which this
                     -- message applies and should be presented
                     -- to the driver
   area    CHOICE  {
      shapePointSet  ShapePointSet,
                     -- A short road segment
      circle         Circle,
                     -- A point and radius
      regionPointSet RegionPointSet
                     -- Wide area enclosed regions
      }
   }


-- DF_VehicleComputedLane (Desc Name) Record 65
VehicleComputedLane ::=  SEQUENCE {
   laneNumber       LaneNumber,
   laneWidth        LaneWidth  OPTIONAL,
   laneAttributes   VehicleLaneAttributes  OPTIONAL,
                    -- if not present, same as ref lane
   refLaneNum       LaneNumber,
                    -- number of the ref lane to be used
   lineOffset       DrivenLineOffset,
   keepOutList      NodeList OPTIONAL,
                    -- no stop points along the path
   connectsTo       ConnectsTo  OPTIONAL,
                    -- a list of other lanes and their
                    -- turning use by this lane
   ...
   }
```

```
-- DF_VehicleIdent (Desc Name) Record 66
VehicleIdent ::=  SEQUENCE {
   name           DescriptiveName OPTIONAL,
                  -- a human readable name for debugging use
   vin            VINstring OPTIONAL,
                  -- vehicle VIN value
   ownerCode      IA5String(SIZE(1..32)) OPTIONAL,
                  -- vehicle owner code
   id             TemporaryID OPTIONAL,
                  -- same value used in the BSM

   vehicleType    VehicleType  OPTIONAL,
   vehicleClass   CHOICE
                  {
                  vGroup ITIS.VehicleGroupAffected,
                  rGroup ITIS.ResponderGroupAffected,
                  rEquip ITIS.IncidentResponseEquipment
                  } OPTIONAL,
   ... -- # LOCAL_CONTENT
   }


-- DF_VehicleReferenceLane (Desc Name) Record 67
VehicleReferenceLane ::=  SEQUENCE {
   laneNumber       LaneNumber,
   laneWidth        LaneWidth  OPTIONAL,
   laneAttributes   VehicleLaneAttributes,
   nodeList         NodeList,
                    -- path details of the lane and width
   keepOutList      NodeList OPTIONAL,
                    -- no stop points along the path
   connectsTo       ConnectsTo  OPTIONAL,
                    -- a list of other lanes and their
                    -- turning use by this lane
   ...
   }


-- DF_VehicleSafetyExtension (Desc Name) Record 68
VehicleSafetyExtension ::=  SEQUENCE {
   events             EventFlags     OPTIONAL,
   pathHistory        PathHistory    OPTIONAL,
   pathPrediction     PathPrediction OPTIONAL,
   theRTCM            RTCMPackage    OPTIONAL,
   ... -- # LOCAL_CONTENT
   }


-- DF_VehicleSize (Desc Name) Record 69
VehicleSize ::=  SEQUENCE {
   width     VehicleWidth,
   length    VehicleLength
   }  -- 3 bytes in length


-- DF_VehicleStatus (Desc Name) Record 70
VehicleStatus ::=  SEQUENCE {
   lights          ExteriorLights OPTIONAL,                -- Exterior Lights
   lightBar        LightbarInUse  OPTIONAL,                -- PS Lights

   wipers   SEQUENCE {
         statusFront    WiperStatusFront,
         rateFront      WiperRate,
         statusRear     WiperStatusRear       OPTIONAL,
         rateRear       WiperRate             OPTIONAL
         } OPTIONAL,                                       -- Wipers

   brakeStatus  BrakeSystemStatus OPTIONAL,
                  -- 2 bytes with the following in it:
                  --   wheelBrakes        BrakeAppliedStatus,
                  --                      -x- 4 bits
                  --   traction           TractionControlState,
                  --                      -x- 2 bits
                  --   abs                AntiLockBrakeStatus,
                  --                      -x- 2 bits
                  --   scs                StablityControlStatus,
                  --                      -x- 2 bits
```

```
                --   brakeBoost         BrakeBoostApplied,
                --                      -x- 2 bits
                --   spareBits
                --                      -x- 4 bits
                -- Note that is present in BSM Part I
                                                        -- Braking Data
   brakePressure   BrakeAppliedPressure  OPTIONAL,      -- Braking Pressure
   roadFriction    CoefficientOfFriction OPTIONAL,      -- Roadway Friction


   sunData         SunSensor             OPTIONAL,      -- Sun Sensor
   rainData        RainSensor            OPTIONAL,      -- Rain Sensor
   airTemp         AmbientAirTemperature OPTIONAL,      -- Air Temperature
   airPres         AmbientAirPressure    OPTIONAL,      -- Air Pressure

   steering   SEQUENCE {
         angle      SteeringWheelAngle,
         confidence SteeringWheelAngleConfidence   OPTIONAL,
         rate       SteeringWheelAngleRateOfChange OPTIONAL,
         wheels     DrivingWheelAngle              OPTIONAL
         } OPTIONAL,                                  -- steering data

   accelSets  SEQUENCE {
         accel4way       AccelerationSet4Way           OPTIONAL,
         vertAccelThres  VerticalAccelerationThreshold OPTIONAL,
                                                       -- Wheel Exceeded point
         yawRateCon      YawRateConfidence             OPTIONAL,
                                                        -- Yaw Rate Confidence
         hozAccelCon     AccelerationConfidence        OPTIONAL,
                                                        -- Acceleration Confidence
         confidenceSet   ConfidenceSet                 OPTIONAL
                                                       -- general ConfidenceSet
         } OPTIONAL,


   object     SEQUENCE {
         obDist          ObstacleDistance,          -- Obstacle Distance
         obDirect        ObstacleDirection,         -- Obstacle Direction
         dateTime        DDateTime                  -- time detected
         } OPTIONAL,                                      -- detected Obstacle data



   fullPos         FullPositionVector OPTIONAL,     -- complete set of time and
                                                    -- position, speed, heading
   throttlePos     ThrottlePosition OPTIONAL,
   speedHeadC      SpeedandHeadingandThrottleConfidence OPTIONAL,
   speedC          SpeedConfidence OPTIONAL,

   vehicleData   SEQUENCE {
         height        VehicleHeight,
         bumpers       BumperHeights,
         mass          VehicleMass,
         trailerWeight TrailerWeight,
         type          VehicleType
         -- values for width and length are sent in BSM part I as well.
         } OPTIONAL,                                 -- vehicle data

   vehicleIdent   VehicleIdent OPTIONAL,             -- comm vehicle data

   j1939data      J1939data OPTIONAL,           -- Various SAE J1938 data items

   weatherReport SEQUENCE {
         isRaining        NTCIP.EssPrecipYesNo,
         rainRate         NTCIP.EssPrecipRate       OPTIONAL,
         precipSituation  NTCIP.EssPrecipSituation  OPTIONAL,
         solarRadiation   NTCIP.EssSolarRadiation   OPTIONAL,
         friction         NTCIP.EssMobileFriction   OPTIONAL
         } OPTIONAL,                                 -- local weather data

   gpsStatus      GPSstatus          OPTIONAL,       -- vehicle's GPS

   ... -- # LOCAL_CONTENT OPTIONAL,
   }


-- DF_VehicleStatusRequest (Desc Name) Record 71
VehicleStatusRequest ::=  SEQUENCE {
   dataType             VehicleStatusDeviceTypeTag,
   subType              INTEGER (1..15)   OPTIONAL,
   sendOnLessThenValue  INTEGER (-32767..32767) OPTIONAL,
   sendOnMoreThenValue  INTEGER (-32767..32767) OPTIONAL,
```

```
   sendAll              BOOLEAN OPTIONAL,
   ...
   }



-- DF_WiperStatus (Desc Name) Record 72
WiperStatus ::=  SEQUENCE {
   statusFront    WiperStatusFront,
   rateFront      WiperRate,
   statusRear     WiperStatusRear      OPTIONAL,
   rateRear       WiperRate            OPTIONAL
   }



-- DF_ITIS_Phrase_WorkZone (Desc Name) Record 73
WorkZone ::=  SEQUENCE (SIZE(1..10)) OF SEQUENCE {
  item CHOICE    {
       itis   ITIS.ITIScodes,
       text   IA5String  (SIZE(1..16))
       } -- # UNTAGGED
  }


-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--
-- Start of entries from table Data_Elements...
-- This table typicaly contains data element entries.
-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--


-- DE_Acceleration (Desc Name) Record 1
Acceleration ::= INTEGER (-2000..2001)
    -- LSB units are 0.01 m/s^2
   -- the value 2000 shall be used for values greater than 2000
   -- the value -2000 shall be used for values less than -2000
   -- a value of 2001 shall be used for Unavailable



-- DE_AccelerationConfidence (Desc Name) Record 2
AccelerationConfidence ::= ENUMERATED {
   unavailable  (0), -- B'000  Not Equipped or data is unavailable
   accl-100-00  (1), -- B'001  100  meters / second squared
   accl-010-00  (2), -- B'010  10   meters / second squared
   accl-005-00  (3), -- B'011  5    meters / second squared
   accl-001-00  (4), -- B'100  1    meters / second squared
   accl-000-10  (5), -- B'101  0.1  meters / second squared
   accl-000-05  (6), -- B'110  0.05 meters / second squared
   accl-000-01  (7)  -- B'111  0.01 meters / second squared
   }
   -- Encoded as a 3 bit value



-- DE_AmbientAirPressure (Barometric Pressure) (Desc Name) Record 3
AmbientAirPressure ::= INTEGER (0..255)
   -- 8 Bits in hPa starting at 580 with a resolution of
   -- 2 hPa resulting in a range of 580 to 1090



-- DE_AmbientAirTemperature (Desc Name) Record 4
AmbientAirTemperature ::= INTEGER (0..191) -- in deg C with a -40 offset



-- DE_AntiLockBrakeStatus (Desc Name) Record 5
AntiLockBrakeStatus ::= ENUMERATED {
   unavailable (0), -- B'00  Vehicle Not Equipped with ABS
                    -- or ABS status is unavailable
   off         (1), -- B'01  Vehicle's ABS is Off
   on          (2), -- B'10  Vehicle's ABS is On (but not engaged)
   engaged     (3)  -- B'11  Vehicle's ABS is Engaged
   }
   -- Encoded as a 2 bit value



-- DE_ApproachNumber (Desc Name) Record 6
ApproachNumber ::= INTEGER (0..127)
```

```
-- DE_AuxiliaryBrakeStatus (Desc Name) Record 7
AuxiliaryBrakeStatus ::= ENUMERATED {
   unavailable (0), -- B'00  Vehicle Not Equipped with Aux Brakes
                    -- or Aux Brakes status is unavailable
   off         (1), -- B'01  Vehicle's Aux Brakes are Off
   on          (2), -- B'10  Vehicle's Aux Brakes are On ( Engaged )
   reserved    (3)  -- B'11
   }
   -- Encoded as a 2 bit value


-- DE_J1939-71-Axle Location (Desc Name) Record 8
AxleLocation ::= INTEGER (0..127)


-- DE_J1939-71-Axle Weight (Desc Name) Record 9
AxleWeight ::= INTEGER (0..65535)


-- DE_BarrierAttributes (Desc Name) Record 10
BarrierAttributes ::= INTEGER (0..8192)
-- With bits as defined:
   noData                   BarrierAttributes ::= 0
                            -- ('0000-0000-0000-0000'B)
   median                   BarrierAttributes ::= 1
                            -- ('0000-0000-0000-0001'B)
   whiteLine                BarrierAttributes ::= 2
                            -- ('0000-0000-0000-0010'B)
   strippedLines            BarrierAttributes ::= 4
                            -- ('0000-0000-0000-0100'B)
   doubleStrippedLines      BarrierAttributes ::= 8
                            -- ('0000-0000-0000-1000'B)
   trafficCones             BarrierAttributes ::= 16
                            -- ('0000-0000-0001-0000'B)
   constructionBarrier      BarrierAttributes ::= 32
                            -- ('0000-0000-0010-0000'B)
   trafficChannels          BarrierAttributes ::= 64
                            -- ('0000-0000-0100-0000'B)
   noCurbs                  BarrierAttributes ::= 128
                            -- ('0000-0000-1000-0000'B)
   lowCurbs                 BarrierAttributes ::= 256
                            -- ('0000-0000-1000-0000'B)
   highCurbs                BarrierAttributes ::= 512
                            -- ('0000-0001-0000-0000'B)
   hovDoNotCross            BarrierAttributes ::= 1024
                            -- ('0000-0010-0000-0000'B)
   hovEntryAllowed          BarrierAttributes ::= 2048
                            -- ('0000-0100-0000-0000'B)
   hovExitAllowed           BarrierAttributes ::= 4096
                            -- ('0000-1000-0000-0000'B)


-- DE_BrakeAppliedPressure (Desc Name) Record 11
BrakeAppliedPressure ::= ENUMERATED {
   unavailable  (0),  -- B'0000 Not Equipped
                      -- or Brake Pres status is unavailable
   minPressure  (1),  -- B'0001 Minimum Braking Pressure
   bkLvl-2      (2),  -- B'0010
   bkLvl-3      (3),  -- B'0011
   bkLvl-4      (4),  -- B'0100
   bkLvl-5      (5),  -- B'0101
   bkLvl-6      (6),  -- B'0110
   bkLvl-7      (7),  -- B'0111
   bkLvl-8      (8),  -- B'1000
   bkLvl-9      (9),  -- B'1001
   bkLvl-10     (10), -- B'1010
   bkLvl-11     (11), -- B'1011
   bkLvl-12     (12), -- B'1100
   bkLvl-13     (13), -- B'1101
   bkLvl-14     (14), -- B'1110
   maxPressure  (15)  -- B'1111 Maximum Braking Pressure
   }
   -- Encoded as a 4 bit value
```

```
-- DE_BrakeAppliedStatus (Desc Name) Record 12
BrakeAppliedStatus ::= BIT STRING {
   allOff      (0), -- B'0000  The condition All Off
   leftFront   (1), -- B'0001  Left Front Active
   leftRear    (2), -- B'0010  Left Rear Active
   rightFront  (4), -- B'0100  Right Front Active
   rightRear   (8)  -- B'1000  Right Rear Active
   } -- to fit in 4 bits


-- DE_BrakeBoostApplied (Desc Name) Record 13
BrakeBoostApplied ::= ENUMERATED {
   unavailable   (0), -- Vehicle not equipped with brake boost
                      -- or brake boost data is unavailable
   off           (1), -- Vehicle's brake boost is off
   on            (2)  -- Vehicle's brake boost is on (applied)
   }
   -- Encoded as a 2 bit value


-- DE_BumperHeightFront (Desc Name) Record 14
BumperHeightFront ::= INTEGER (0..127) -- in units of 0.01 meters from ground surface.


-- DE_BumperHeightRear (Desc Name) Record 15
BumperHeightRear ::= INTEGER (0..127) -- in units of 0.01 meters from ground surface.


-- DE_J1939-71-Cargo Weight (Desc Name) Record 16
CargoWeight ::= INTEGER (0..65535)


-- DE_CodeWord (Desc Name) Record 17
CodeWord ::= OCTET STRING (SIZE(1..16))
   -- any octect string up to 16 bytes


-- DE_CoefficientOfFriction (Desc Name) Record 18
CoefficientOfFriction ::= INTEGER (0..50)
   -- where 0 = 0.00 micro (frictonless)
   -- and 50 = 0.98 micro, in steps of 0.02


-- DE_ColorState (Desc Name) Record 19
ColorState ::= ENUMERATED {
   dark            (0),  -- (B0000) Dark, lights inactive
   green           (1),  -- (B0001)
   green-flashing  (9),  -- (B1001)

   yellow          (2),  -- (B0010)
   yellow-flashing (10), -- (B1010)

   red             (4),  -- (B0100)
   red-flashing    (12)  -- (B1100)

   } -- a 4 bit encoded value
     -- note that above may be combined
     -- to create additonal patterns


-- DE_Count (Desc Name) Record 20
Count ::= INTEGER (0..32)


-- DE_CrosswalkLaneAttributes (Desc Name) Record 21
CrosswalkLaneAttributes ::= ENUMERATED {
   noData                    (0),   -- ('0000000000000000'B)
   twoWayPath                (1),   -- ('0000000000000001'B)
   pedestrianCrosswalk       (2),   -- ('0000000000000010'B)
   bikeLane                  (4),   -- ('0000000000000100'B)
   railRoadTrackPresent      (8),   -- ('0000000000001000'B)
   oneWayPathOfTravel        (16),  -- ('0000000000010000'B)
   pedestrianCrosswalkTypeA  (32),  -- ('0000000000100000'B)
   pedestrianCrosswalkTypeB  (64),  -- ('0000000001000000'B)
```

```
   pedestrianCrosswalkTypeC   (128)  -- ('0000000010000000'B)
   }
   -- MUTCD provides no real "types" to use here


-- DE_DDay (Desc Name) Record 22
DDay ::= INTEGER (0..31)  -- units of days


-- DE_DescriptiveName (Desc Name) Record 23
DescriptiveName ::= IA5String (SIZE(1..63))


-- DE_DHour (Desc Name) Record 24
DHour ::= INTEGER (0..31) -- units of hours


-- DE_DirectionOfUse (Desc Name) Record 25
DirectionOfUse ::= ENUMERATED {
   forward    (0), -- direction of travel follows node ordering
   reverse    (1), -- direction of travel is the reverse of node ordering
   both       (2), -- direction of travel allowed in both directions
   ...
   }


-- DE_DMinute (Desc Name) Record 26
DMinute ::= INTEGER (0..63) -- units of minutes


-- DE_DMonth (Desc Name) Record 27
DMonth ::= INTEGER (0..15) -- units of months


-- DE_DOffset (Desc Name) Record 28
DOffset ::= INTEGER (-840..840) -- units of minutes from UTC time


-- DE_J1939-71-Drive Axle Lift Air Pressure (Desc Name) Record 29
DriveAxleLiftAirPressure ::= INTEGER (0..1000)


-- DE_J1939-71-Drive Axle Location (Desc Name) Record 30
DriveAxleLocation ::= INTEGER (0..255)


-- DE_J1939-71-Drive Axle Lube Pressure (Desc Name) Record 31
DriveAxleLubePressure ::= INTEGER (0..1000)


-- DE_J1939-71-Drive Axle Temperature (Desc Name) Record 32
DriveAxleTemperature ::= INTEGER (-40..210)


-- DE_DrivenLineOffset (Desc Name) Record 33
DrivenLineOffset ::= INTEGER (-32767..32767)
   -- LSB units are 1 cm.


-- DE_DrivingWheelAngle (Desc Name) Record 34
DrivingWheelAngle ::= INTEGER (-127..127)
   -- LSB units of 0.3333 degrees.
   -- a range of 42.33 degrees each way


-- DE_DSecond (Desc Name) Record 35
DSecond ::= INTEGER (0..65535) -- units of miliseconds
```

```
-- DE_DSignalSeconds (Desc Name) Record 36
DSignalSeconds ::= INTEGER (0..30000) -- units of 0.01 seconds


-- DE_DSRC_MessageID (Desc Name) Record 37
DSRCmsgID ::= ENUMERATED {
   reserved                        (0),
   alaCarteMessage                 (1),  -- ACM
   basicSafetyMessage              (2),  -- BSM, heartbeat msg
   basicSafetyMessageVerbose       (3),  -- used for testing only
   commonSafetyRequest             (4),  -- CSR
   emergencyVehicleAlert           (5),  -- EVA
   intersectionCollisionAlert      (6),  -- ICA
   mapData                         (7),  -- MAP, GID, intersections
   nmeaCorrections                 (8),  -- NMEA
   probeDataManagement             (9),  -- PDM
   probeVehicleData                (10), -- PVD
   roadSideAlert                   (11), -- RSA
   rtcmCorrections                 (12), -- RTCM
   signalPhaseAndTimingMessage     (13), -- SPAT
   signalRequestMessage            (14), -- SRM
   signalStatusMessage             (15), -- SSM
   travelerInformation             (16), -- TIM

   ... -- # LOCAL_CONTENT
   }
   -- values to 127 reserved for std use
   -- values 128 to 255 reserved for local use


-- DE_DYear (Desc Name) Record 38
DYear ::= INTEGER (0..9999) -- units of years


-- DE_Elevation (Desc Name) Record 39
Elevation ::= OCTET STRING (SIZE(2))
   -- 1 decimeter LSB (10 cm)
   -- Encode elevations from 0 to 6143.9 meters
   -- above the reference ellipsoid as 0x0000 to 0xEFFF.
   -- Encode elevations from -409.5 to -0.1 meters,
   -- i.e. below the reference ellipsoid, as 0xF001 to 0xFFFF
   -- unknown as 0xF000


-- DE_ElevationConfidence (Desc Name) Record 40
ElevationConfidence ::= ENUMERATED {
   unavailable (0),  -- B'0000  Not Equipped or unavailable
   elev-500-00 (1),  -- B'0001  (500 m)
   elev-200-00 (2),  -- B'0010  (200 m)
   elev-100-00 (3),  -- B'0011  (100 m)
   elev-050-00 (4),  -- B'0100  (50 m)
   elev-020-00 (5),  -- B'0101  (20 m)
   elev-010-00 (6),  -- B'0110  (10 m)
   elev-005-00 (7),  -- B'0111  (5 m)
   elev-002-00 (8),  -- B'1000  (2 m)
   elev-001-00 (9),  -- B'1001  (1 m)
   elev-000-50 (10), -- B'1010  (50 cm)
   elev-000-20 (11), -- B'1011  (20 cm)
   elev-000-10 (12), -- B'1100  (10 cm)
   elev-000-05 (13), -- B'1101  (5 cm)
   elev-000-02 (14), -- B'1110  (2 cm)
   elev-000-01 (15)  -- B'1111  (1 cm)
   }
   -- Encoded as a 4 bit value


-- DE_EmergencyDetails (Desc Name) Record 41
EmergencyDetails ::= INTEGER (0..63)
   -- First two bit (MSB set to zero.
   -- Combining these 3 items in the remaning 6 bits
   -- sirenUse        SirenInUse
   -- lightsUse       LightbarInUse
   -- multi           MultiVehicleReponse


-- DE_EventFlags (Desc Name) Record 42
```

```
EventFlags ::= INTEGER (0..8192)
-- With bits as defined:
   eventHazardLights               EventFlags ::= 1
   eventStopLineViolation          EventFlags ::= 2 -- Intersection Violation
   eventABSactivated               EventFlags ::= 4
   eventTractionControlLoss        EventFlags ::= 8
   eventStabilityControlactivated  EventFlags ::= 16
   eventHazardousMaterials         EventFlags ::= 32
   eventEmergencyResponse          EventFlags ::= 64
   eventHardBraking                EventFlags ::= 128
   eventLightsChanged              EventFlags ::= 256
   eventWipersChanged              EventFlags ::= 512
   eventFlatTire                   EventFlags ::= 1024
   eventDisabledVehicle            EventFlags ::= 2048
   eventAirBagDeployment           EventFlags ::= 4096


-- DE_Extent (Desc Name) Record 43
Extent ::= ENUMERATED {
    useInstantlyOnly   (0),
    useFor3meters      (1),
    useFor10meters     (2),
    useFor50meters     (3),
    useFor100meters    (4),
    useFor500meters    (5),
    useFor1000meters   (6),
    useFor5000meters   (7),
    useFor10000meters  (8),
    useFor50000meters  (9),
    useFor100000meters (10),
    forever            (127)  -- very wide area
    }
    -- encode as a single byte


-- DE_ExteriorLights (Desc Name) Record 44
ExteriorLights ::= INTEGER (0..256)
-- With bits as defined:
   allLightsOff              ExteriorLights ::= 0
                             -- B'0000-0000
   lowBeamHeadlightsOn       ExteriorLights ::= 1
                             -- B'0000-0001
   highBeamHeadlightsOn      ExteriorLights ::= 2
                             -- B'0000-0010
   leftTurnSignalOn          ExteriorLights ::= 4
                             -- B'0000-0100
   rightTurnSignalOn         ExteriorLights ::= 8
                             -- B'0000-1000
   hazardSignalOn            ExteriorLights ::= 12
                             -- B'0000-1100
   automaticLightControlOn   ExteriorLights ::= 16
                             -- B'0001-0000
   daytimeRunningLightsOn    ExteriorLights ::= 32
                             -- B'0010-0000
   fogLightOn                ExteriorLights ::= 64
                             -- B'0100-0000
   parkingLightsOn           ExteriorLights ::= 128
                             -- B'1000-0000


-- DE_FurtherInfoID (Desc Name) Record 45
FurtherInfoID ::= OCTET STRING (SIZE(2))
   -- a link to any other incident
   -- information data that may be available
   -- in the normal ATIS incident description
   -- or other messages
   -- two value bytes in length


-- DE_GPSstatus (Desc Name) Record 46
GPSstatus ::= BIT STRING {
   unavailable               (0), -- Not Equipped or unavailable
   isHealthy                 (1),
   isMonitored               (2),
   baseStationType           (3), -- Set to zero if a moving base station,
                                  -- set to one if it is a fixed base station
   aPDOPofUnder5             (4), -- A dilution of precision greater then 5
   inViewOfUnder5            (5), -- Less then 5 satellites in view
   localCorrectionsPresent   (6),
```

```
   networkCorrectionsPresent (7)
   } -- (SIZE(1))


-- DE_Heading (Desc Name) Record 47
Heading ::= INTEGER (0..28800)
   -- LSB of 0.0125 degrees
   -- A range of 0 to 359.9875 degrees


-- DE_HeadingConfidence (Desc Name) Record 48
HeadingConfidence ::= ENUMERATED {
   unavailable (0), -- B'000  Not Equipped or unavailable
   prec45deg   (1), -- B'001  45   degrees
   prec10deg   (2), -- B'010  10   degrees
   prec05deg   (3), -- B'011  5    degrees
   prec01deg   (4), -- B'100  1    degrees
   prec0-1deg  (5), -- B'101  0.1  degrees
   prec0-05deg (6), -- B'110  0.05 degrees
   prec0-01deg (7)  -- B'111  0.01 degrees
   }
   -- Encoded as a 3 bit value


-- DE_HeadingSlice (Desc Name) Record 49
HeadingSlice ::= OCTET STRING (SIZE(2))
   -- Each bit 22.5 degree starting from
   -- North and moving Eastward (clockwise)

   --  Define global enums for this entry
   noHeading               HeadingSlice ::= '0000'H
   allHeadings             HeadingSlice ::= 'FFFF'H

   from000-0to022-5degrees  HeadingSlice ::= '0001'H
   from022-5to045-0degrees  HeadingSlice ::= '0002'H
   from045-0to067-5degrees  HeadingSlice ::= '0004'H
   from067-5to090-0degrees  HeadingSlice ::= '0008'H

   from090-0to112-5degrees  HeadingSlice ::= '0010'H
   from112-5to135-0degrees  HeadingSlice ::= '0020'H
   from135-0to157-5degrees  HeadingSlice ::= '0040'H
   from157-5to180-0degrees  HeadingSlice ::= '0080'H

   from180-0to202-5degrees  HeadingSlice ::= '0100'H
   from202-5to225-0degrees  HeadingSlice ::= '0200'H
   from225-0to247-5degrees  HeadingSlice ::= '0400'H
   from247-5to270-0degrees  HeadingSlice ::= '0800'H

   from270-0to292-5degrees  HeadingSlice ::= '1000'H
   from292-5to315-0degrees  HeadingSlice ::= '2000'H
   from315-0to337-5degrees  HeadingSlice ::= '4000'H
   from337-5to360-0degrees  HeadingSlice ::= '8000'H


-- DE_IntersectionID (Desc Name) Record 50
IntersectionID ::= OCTET STRING (SIZE(2..4))
   -- note that often only the lower 16 bits of this value
   -- will be sent as the operational region (state etc) will
   -- be known and not sent each time


-- DE_Intersection Status Object (Desc Name) Record 51
IntersectionStatusObject ::= OCTET STRING (SIZE(1))
   -- with bits set as follows Bit #:
   -- 0    Manual Control is enabled.  Timing reported is per
   --      programmed values, etc but person at cabinet can
   --      manually request that certain intervals are terminated
   --      early (e.g. green).
   -- 1    Stop Time is activated and all counting/timing has stopped.
   -- 2    Intersection is in Conflict Flash.
   -- 3    Preempt is Active
   -- 4    Transit Signal Priority (TSP) is Active
   -- 5    Reserved
   -- 6    Reserved
   -- 7    Reserved as zero
```

```
-- DE_LaneCount (Desc Name) Record 52
LaneCount ::= INTEGER (0..255) -- the number of lanes to follow


-- DE_LaneManeuverCode (Desc Name) Record 53
LaneManeuverCode ::= ENUMERATED {
   unknown          (0),  -- used for N.A. as well
   uTurn            (1),
   leftTurn         (2),
   rightTurn        (3),
   straightAhead    (4),
   softLeftTurn     (5),
   softRightTurn    (6),
   ...
   }
   -- values to 127 reserved for std use
   -- values 128 to 255 reserved for local use


-- DE_LaneNumber (Desc Name) Record 54
LaneNumber ::= OCTET STRING (SIZE(1))


-- DE_LaneSet (Desc Name) Record 55
LaneSet ::= OCTET STRING (SIZE(1..127))
    -- each byte encoded as a: LaneNumber,
    -- the collection of lanes, by num,
    -- to which some state data applies


-- DE_LaneWidth (Desc Name) Record 56
LaneWidth ::= INTEGER (0..32767) -- units of 1 cm


-- DE_Latitude (Desc Name) Record 57
Latitude ::= INTEGER (-900000000..900000001)
   -- LSB = 1/10 micro degree
   -- Providing a range of plus-minus 90 degrees


-- DE_LayerID (Desc Name) Record 58
LayerID ::= INTEGER (0..100)


-- DE_LayerType (Desc Name) Record 59
LayerType ::= ENUMERATED {
     none                 (0),
     mixedContent         (1), -- two or more of the below types
     generalMapData       (2),
     intersectionData     (3),
     curveData            (4),
     roadwaySectionData   (5),
     parkingAreaData      (6),
     sharedLaneData       (7),
     ... -- # LOCAL_CONTENT
     }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use


-- DE_LightbarInUse (Desc Name) Record 60
LightbarInUse ::= ENUMERATED {
     unavailable         (0),  -- Not Equipped or unavailable
     notInUse            (1),  -- none active
     inUse               (2),
     sirenInUse          (3),
     yellowCautionLights (4),
     schooldBusLights    (5),
     arrowSignsActive    (6),
     slowMovingVehicle   (7),
     freqStops           (8),
     reserved            (9)  -- for future use
     }
```

```
-- DE_MAYDAY_Location_quality_code (Desc Name) Record 61
Location-quality ::= ENUMERATED {
     loc-qual-bt1m       (0), -- quality better than 1 meter
     loc-qual-bt5m       (1), -- quality better than 5 meters
     loc-qual-bt12m      (2), -- quality better than 12.5 meters
     loc-qual-bt50m      (3), -- quality better than 50 meters
     loc-qual-bt125m     (4), -- quality better than 125 meters
     loc-qual-bt500m     (5), -- quality better than 500 meters
     loc-qual-bt1250m    (6), -- quality better than 1250 meters
     loc-qual-unknown    (7)  -- quality value unknown
     } -- 3 bits, appends with loc-tech to make one octet (0..7)


-- DE_MAYDAY_Location_tech_code (Desc Name) Record 62
Location-tech ::= ENUMERATED {
     loc-tech-unknown    (0), -- technology type unknown
     loc-tech-GPS        (1), -- GPS technology only
     loc-tech-DGPS       (2), -- differential GPS (DGPS) technology
     loc-tech-drGPS      (3), -- dead reckoning system w/GPS
     loc-tech-drDGPS     (4), -- dead reckoning system w/DGPS
     loc-tech-dr         (5), -- dead reckoning only
     loc-tech-nav        (6), -- autonomous navigation system on-board
     ...,
     loc-tech-fault      (31) -- feature is not working
     }    -- (0..31) 5 bits, appends with loc-quality to make one octet


-- DE_Longitude (Desc Name) Record 63
Longitude ::= INTEGER (-1800000000..1800000001)
   -- LSB = 1/10 micro degree
   -- Providing a range of plus-minus 180 degrees


-- DE_MinuteOfTheYear (Desc Name) Record 64
MinuteOfTheYear ::= INTEGER (0..525960)


-- DE_MinutesDuration (Desc Name) Record 65
MinutesDuration ::= INTEGER (0..32000) -- units of minutes


-- DE_MsgCount (Desc Name) Record 66
MsgCount ::= INTEGER (0..127)


-- DE_MsgCRC (Desc Name) Record 67
MsgCRC ::= OCTET STRING (SIZE(2)) -- created with the CRC-CCITT polynomial


-- DE_MultiVehicleResponse (Desc Name) Record 68
MultiVehicleResponse ::= ENUMERATED {
     unavailable   (0), -- Not Equipped or unavailable
     singleVehicle (1),
     multiVehicle  (2),
     reserved      (3)  -- for future use
     }


-- DE_MUTCDCode (Desc Name) Record 69
MUTCDCode ::= ENUMERATED {
   none            (0), -- non-MUTCD information
   regulatory      (1), -- "R" Regulatory signs
   warning         (2), -- "W" warning signs
   maintenance     (3), -- "M" Maintenance and construction
   motoristService (4), -- Motorist Services
   guide           (5), -- "G" Guide signs
   rec             (6), -- Recreation and Cultural Interest
   ... -- # LOCAL_CONTENT
   }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use
```

```
-- DE_NMEA_MsgType (Desc Name) Record 70
NMEA-MsgType ::= INTEGER (0..32767)


-- DE_NMEA_Payload (Desc Name) Record 71
NMEA-Payload ::= OCTET STRING (SIZE(1..1023))


-- DE_NMEA_Revision (Desc Name) Record 72
NMEA-Revision ::= ENUMERATED {
     unknown       (0),
     reserved      (1),
     rev1          (10),
     rev2          (20),
     rev3          (30),
     rev4          (40),
     rev5          (50),
     ... -- # LOCAL_CONTENT
     }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use


-- DE_NTCIPVehicleclass, (Desc Name) Record 73
NTCIPVehicleclass ::= OCTET STRING (SIZE(1))
   -- With bits set as per NTCIP values
   -- Priority Request Vehicle Class Type
   -- in the upper nibble
   -- Priority Request Vehicle Class Level
   -- in the lower nibble


-- DE_ObjectCount (Desc Name) Record 74
ObjectCount ::= INTEGER (0..6000) -- a count of objects


-- DE_ObstacleDirection (Desc Name) Record 75
ObstacleDirection ::= Heading -- Use the header DE for this unless it proves different.


-- DE_ObstacleDistance (Desc Name) Record 76
ObstacleDistance ::= INTEGER (0..32767) -- LSB units of meters


-- DE_Payload (Desc Name) Record 77
Payload ::= OCTET STRING (SIZE(1..64))


-- DE_PayloadData (Desc Name) Record 78
PayloadData ::= OCTET STRING (SIZE(1..2048))


-- DE_PedestrianDetect (Desc Name) Record 79
PedestrianDetect ::= ENUMERATED {
   none       (0), -- (B00000001)
   maybe      (1), -- (B00000010)
   one        (2), -- (B00000100)
   some       (3), -- (B00001000) Indicates more than one
   ...
   } -- one byte


-- DE_PedestrianSignalState (Desc Name) Record 80
PedestrianSignalState ::= ENUMERATED {
   unavailable (0), -- Not Equipped or unavailable
   stop        (1), -- (B00000001) do not walk
   caution     (2), -- (B00000010) flashing dont walk sign
   walk        (3), -- (B00000100) walk active
   ...
   } -- one byte
```

```
-- DE_PositionConfidence (Desc Name) Record 81
PositionConfidence ::= ENUMERATED {
   unavailable (0),  -- B'0000  Not Equipped or unavailable
   a500m   (1), -- B'0001  500m  or about 5 * 10 ^ -3 decimal degrees
   a200m   (2), -- B'0010  200m  or about 2 * 10 ^ -3 decimal degrees
   a100m   (3), -- B'0011  100m  or about 1 * 10 ^ -3 decimal degrees
   a50m    (4), -- B'0100  50m   or about 5 * 10 ^ -4 decimal degrees
   a20m    (5), -- B'0101  20m   or about 2 * 10 ^ -4 decimal degrees
   a10m    (6), -- B'0110  10m   or about 1 * 10 ^ -4 decimal degrees
   a5m     (7), -- B'0111  5m    or about 5 * 10 ^ -5 decimal degrees
   a2m     (8), -- B'1000  2m    or about 2 * 10 ^ -5 decimal degrees
   a1m     (9), -- B'1001  1m    or about 1 * 10 ^ -5 decimal degrees
   a50cm  (10), -- B'1010  0.50m or about 5 * 10 ^ -6 decimal degrees
   a20cm  (11), -- B'1011  0.20m or about 2 * 10 ^ -6 decimal degrees
   a10cm  (12), -- B'1100  0.10m or about 1 * 10 ^ -6 decimal degrees
   a5cm   (13), -- B'1101  0.05m or about 5 * 10 ^ -7 decimal degrees
   a2cm   (14), -- B'1110  0.02m or about 2 * 10 ^ -7 decimal degrees
   a1cm   (15)  -- B'1111  0.01m or about 1 * 10 ^ -7 decimal degrees
   }
   -- Encoded as a 4 bit value


-- DE_PreemptState (Desc Name) Record 82
PreemptState ::= ENUMERATED {
   none                       (0), -- No preemption (same as value = 2)
   other                      (1), -- Other
   notActive                  (2), -- Not Active (same as value = 0)
   notActiveWithCall          (3), -- Not Active With Call
   entryStarted               (4), -- Entry Started
   trackService               (5), -- Track Service
   dwell                      (6), -- Dwell
   linkActive                 (7), -- Link Active
   existStarted               (8), -- Exit Started
   maximumPresence            (9), -- Max Presence
   ackowledgedButOverridden (10), -- Ackowledged but Over-ridden
   ... -- # LOCAL_CONTENT
   }
   -- To use 4 bits,
   -- typically packed with other items in a BYTE


-- DE_Priority (Desc Name) Record 83
Priority ::= OCTET STRING (SIZE(1))
    -- Follow definition notes on setting these bits


-- DE_PriorityState (Desc Name) Record 84
PriorityState ::= ENUMERATED {
   noneActive         (0), -- No signal priority (same as value = 1)
   none               (1), -- TSP None
   requested          (2), -- TSP Requested
   active             (3), -- TSP Active
   activeButIhibitd   (4), -- TSP Reservice (active but inhibited)
   seccess            (5), -- TSP Success
   removed            (6), -- TSP Removed
   clearFail          (7), -- TSP Clear Fail
   detectFail         (8), -- TSP Detect Fail
   detectClear        (9), -- TSP Detect Clear
   abort             (10), -- TSP Abort (needed to remain on-line)
   delayTiming       (11), -- TSP Delay Timing
   extendTiming      (12), -- TSP Extend Timing
   preemptOverride   (13), -- TSP Preempt Over-ride
   adaptiveOverride  (14), -- TSP Adaptive Over-ride
   reserved          (15),
   ... -- # LOCAL_CONTENT
   }
   -- To use 4 bits,
   -- typically packed with other items in a BYTE


-- DE_ProbeSegmentNumber (Desc Name) Record 85
ProbeSegmentNumber ::= INTEGER (0..32767)
   -- value determined by local device
   -- as per standard


-- DE_RainSensor (Desc Name) Record 86
RainSensor ::= ENUMERATED {
```

```
     none               (0),
     lightMist          (1),
     heavyMist          (2),
     lightRainOrDrizzle (3),
     rain               (4),
     moderateRain       (5),
     heavyRain          (6),
     heavyDownpour      (7)
     }


-- DE_RequestedItem (Desc Name) Record 87
RequestedItem ::= ENUMERATED {
   reserved      (0),
   itemA         (1),
        -- consisting of 2 elements:
        -- lights          ExteriorLights
        -- lightBar        LightbarInUse

   itemB         (2),
        -- consisting of:
        -- wipers  a SEQUENCE

   itemC         (3),
        -- consisting of:
        -- brakeStatus  BrakeSystemStatus

   itemD         (4),
        -- consisting of 2 elements:
        -- brakePressure   BrakeAppliedPressure
        -- roadFriction    CoefficientOfFriction

   itemE         (5),
        -- consisting of 4 elements:
        -- sunData         SunSensor
        -- rainData        RainSensor
        -- airTemp         AmbientAirTemperature
        -- airPres         AmbientAirPressure

   itemF         (6),
        -- consisting of:
        -- steering  a SEQUENCE

   itemG         (7),
        -- consisting of:
        -- accelSets a SEQUENCE

   itemH         (8),
        -- consisting of:
        -- object a SEQUENCE


   itemI         (9),
        -- consisting of:
        -- fullPos         FullPositionVector

   itemJ         (10),
        -- consisting of:
        -- position2D      Position2D

   itemK         (11),
        -- consisting of:
        -- position3D      Position3D

   itemL         (12),
        -- consisting of 2 elements:
        -- speedHeadC      SpeedandHeadingConfidence
        -- speedC          SpeedConfidence

   itemM         (13),
        -- consisting of:
        -- vehicleData a SEQUENCE

   itemN         (14),
        -- consisting of:
        -- vehicleIdent   VehicleIdent

   itemO         (15),
        -- consisting of:
        -- weatherReport a SEQUENCE

   itemP         (16),
```

```
        -- consisting of:
        -- breadcrumbs    VehicleMotionTrail

   itemQ         (17),
        -- consisting of:
        -- gpsStatus      GPSstatus

   ... -- # LOCAL_CONTENT OPTIONAL,
   }
   -- values to 127 reserved for std use
   -- values 128 to 255 reserved for local use



-- DE_ResponseType (Desc Name) Record 88
ResponseType ::= ENUMERATED {
   notInUseOrNotEquipped      (0),
   emergency                  (1),
   nonEmergency               (2),
   pursuit                    (3)
   -- all others Future Use
   }
   -- values to 127 reserved for std use
   -- values 128 to 255 reserved for local use



-- DE_RTCM_ID (Desc Name) Record 89
RTCM-ID ::= INTEGER (0..32767)



-- DE_RTCM_Payload (Desc Name) Record 90
RTCM-Payload ::= OCTET STRING (SIZE(1..1023))



-- DE_RTCM_Revision (Desc Name) Record 91
RTCM-Revision ::= ENUMERATED {
     unknown       (0),
     reserved      (1),
     rtcmCMR       (2),
     rtcmCMR-Plus  (3),
     rtcmSAPOS     (4),
     rtcmSAPOS-Adv (5),
     rtcmRTCA      (6),
     rtcmRAW       (7),
     rtcmRINEX     (8),
     rtcmSP3       (9),
     rtcmBINEX     (10),
     rtcmRev2-x    (19), -- Used when specific rev is not known
     rtcmRev2-0    (20),
     rtcmRev2-1    (21),
     rtcmRev2-3    (23), -- Std 10402.3
     rtcmRev3-0    (30),
     rtcmRev3-1    (31), -- Std 10403.1
     ... -- # LOCAL_CONTENT
     }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use



-- DE_SignalLightState (Desc Name) Record 92
SignalLightState ::= INTEGER (0..536870912)
  -- The above bit ranges map to each type of direction
  -- using the bits defined by the above table of the standard.



-- DE_SignalReqScheme (Desc Name) Record 93
SignalReqScheme ::= OCTET STRING (SIZE(1))
    -- Encoded as follows:
   -- upper nibble:  Preempt #:
    -- Bit 7 (MSB) 1 =  Preempt and 0 =  Priority
    -- Remaining 3 bits:
    -- Range of 0..7. The values of 1..6 represent
    -- the respective controller preempt or Priority
    -- to be activated. The value of 7 represents a
    -- request for a cabinet flash prempt,
    -- while the value of 0 is reserved.

   -- lower nibble:  Strategy #:
```

```
    -- Range is 0..15 and is used to specify a desired
    -- strategy (if available).
    -- Currently no strategies are defined and this
    -- should be zero.


-- DE_SignalState (Desc Name) Record 94
SignalState ::= OCTET STRING (SIZE(1))
    -- With bits set as follows:

    -- Bit 7 (MSB) Set if the state is currently active
    -- only one active state can exist at a time, and
    -- this state should be sent first in any sequences

    -- Bits 6~4 The preempt or priority value that is
    -- being described.

    -- Bits 3~0 the state bits, indicating either a
    -- preemption or a priority use as follows:

    -- If a preemption: to follow the
    -- preemptState object of NTCIP 1202 v2.19f
    -- See PreemptState for bit definitions.

    -- If a prioirty to follow the
    -- tspInputStatus object utilized in the
    -- NYC ASTC2 traffic controller
    -- See PriorityState for bit definitions


-- DE_SignPrority (Desc Name) Record 95
SignPrority ::= INTEGER (0..7)
   -- 0 as least, 7 as most


-- DE_SirenInUse (Desc Name) Record 96
SirenInUse ::= ENUMERATED {
     unavailable   (0), -- Not Equipped or unavailable
     notInUse      (1),
     inUse         (2),
     reserved      (3)  -- for future use
     }


-- DE_SpecialLaneAttributes (Desc Name) Record 97
SpecialLaneAttributes ::= ENUMERATED {
   noData                    (0), -- ('0000000000000000'B)
   egressPath                (1), -- ('0000000000000001'B)
       -- a two-way path or an outbound path is described
   railRoadTrack             (2), -- ('0000000000000010'B)
   transitOnlyLane           (4), -- ('0000000000000100'B)
   hovLane                   (8), -- ('0000000000001000'B)
   busOnly                  (16), -- ('0000000000010000'B)
   vehiclesEntering         (32), -- ('0000000000100000'B)
   vehiclesLeaving          (64), -- ('0000000001000000'B)
   reserved                (128)  -- ('0000000010000000'B)
   } -- 1 byte


-- DE_SpecialSignalState (Desc Name) Record 98
SpecialSignalState ::= ENUMERATED {
   unknown    (0),
   notInUse   (1), -- (B0001) default state, empty, not in use
   arriving   (2), -- (B0010) track-lane about to be occupied
   present    (3), -- (B0100) track-lane is occupied with vehicle
   departing  (4), -- (B1000) track-lane about to be empty
   ...
   } -- one byte


-- DE_Speed (Desc Name) Record 99
Speed ::= INTEGER (0..8191) -- Units of 0.02 m/s
          -- The value 8191 indicates that
          -- speed is unavailable
```

```
-- DE_SpeedConfidence (Desc Name) Record 100
SpeedConfidence ::= ENUMERATED {
   unavailable (0), -- B'000  Not Equipped or unavailable
   prec100ms   (1), -- B'001  100  meters / sec
   prec10ms    (2), -- B'010  10   meters / sec
   prec5ms     (3), -- B'011  5    meters / sec
   prec1ms     (4), -- B'100  1    meters / sec
   prec0-1ms   (5), -- B'101  0.1  meters / sec
   prec0-05ms  (6), -- B'110  0.05 meters / sec
   prec0-01ms  (7)  -- B'111  0.01 meters / sec
   }
   -- Encoded as a 3 bit value


-- DE_StabilityControlStatus (Desc Name) Record 101
StabilityControlStatus ::= ENUMERATED {
   unavailable (0), -- B'00  Not Equipped with SC
                    -- or SC status is unavailable
   off         (1), -- B'01  Off
   on          (2)  -- B'10  On or active (engaged)
   }
   -- Encoded as a 2 bit value


-- DE_StateConfidence (Desc Name) Record 102
StateConfidence ::= ENUMERATED {
   unKnownEstimate        (0),
   minTime                (1),
   maxTime                (2),
   timeLikeklyToChange    (3),
   ... -- # LOCAL_CONTENT
   }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use


-- DE_J1939-71-Steering Axle Lube Pressure (Desc Name) Record 103
SteeringAxleLubePressure ::= INTEGER (0..255)


-- DE_J1939-71-Steering Axle Temperature (Desc Name) Record 104
SteeringAxleTemperature ::= INTEGER (0..255)


-- DE_SteeringWheelAngle (Desc Name) Record 105
SteeringWheelAngle ::= OCTET STRING (SIZE(1))
    -- LSB units of 1.5 degrees.
    -- a range of -189 to +189 degrees
    -- 0x01 = 00 = +1.5 deg
    -- 0x81 = -126 = -189 deg and beyond
    -- 0x7E = +126 = +189 deg and beyond
    -- 0x7F = +127 to be used for unavailable


-- DE_SteeringWheelAngleConfidence (Desc Name) Record 106
SteeringWheelAngleConfidence ::= ENUMERATED {
   unavailable (0), -- B'00  Not Equipped with Wheel angle
                    -- or Wheel angle status is unavailable
   prec2deg    (1), -- B'01  2 degrees
   prec1deg    (2), -- B'10  1 degree
   prec0-02deg (3)  -- B'11  0.02 degrees
   }
   -- Encoded as a 2 bit value


-- DE_SteeringWheelAngleRateOfChange (Desc Name) Record 107
SteeringWheelAngleRateOfChange ::= INTEGER (-127..127)
   -- LSB is 3 degrees per second


-- DE_SunSensor (Desc Name) Record 108
SunSensor ::= INTEGER (0..1000)
   -- units of watts / m2
```

```
-- DE_TemporaryID (Desc Name) Record 109
TemporaryID ::= OCTET STRING (SIZE(4)) -- a 4 byte string array


-- DE_TerminationDistance (Desc Name) Record 110
TermDistance ::= INTEGER (1..30000)   -- units in meters


-- DE_TerminationTime (Desc Name) Record 111
TermTime ::= INTEGER (1..1800) -- units of sec


-- DE_ThrottleConfidence (Desc Name) Record 112
ThrottleConfidence ::= ENUMERATED {
   unavailable     (0), -- B'00  Not Equipped or unavailable
   prec10percent   (1), -- B'01  10  percent Confidence level
   prec1percent    (2), -- B'10  1   percent Confidence level
   prec0-5percent  (3)  -- B'11  0.5 percent Confidence level
   }
   -- Encoded as a 2 bit value


-- DE_ThrottlePosition (Desc Name) Record 113
ThrottlePosition ::= INTEGER (0..200) -- LSB units are 0.5 percent


-- DE_TimeConfidence (Desc Name) Record 114
TimeConfidence ::= ENUMERATED {
   unavailable              (0), -- Not Equipped or unavailable
   time-100-000             (1), -- Better then  100 Seconds
   time-050-000             (2), -- Better then  50 Seconds
   time-020-000             (3), -- Better then  20 Seconds
   time-010-000             (4), -- Better then  10 Seconds
   time-002-000             (5), -- Better then  2 Seconds
   time-001-000             (6), -- Better then  1 Second
   time-000-500             (7), -- Better then  0.5 Seconds
   time-000-200             (8), -- Better then  0.2 Seconds
   time-000-100             (9), -- Better then  0.1 Seconds
   time-000-050            (10), -- Better then  0.05 Seconds
   time-000-020            (11), -- Better then  0.02 Seconds
   time-000-010            (12), -- Better then  0.01 Seconds
   time-000-005            (13), -- Better then  0.005 Seconds
   time-000-002            (14), -- Better then  0.002 Seconds
   time-000-001            (15), -- Better then  0.001 Seconds
                                 -- Better then one milisecond
   time-000-000-5          (16), -- Better then  0.000,5 Seconds
   time-000-000-2          (17), -- Better then  0.000,2 Seconds
   time-000-000-1          (18), -- Better then  0.000,1 Seconds
   time-000-000-05         (19), -- Better then  0.000,05 Seconds
   time-000-000-02         (20), -- Better then  0.000,02 Seconds
   time-000-000-01         (21), -- Better then  0.000,01 Seconds
   time-000-000-005        (22), -- Better then  0.000,005 Seconds
   time-000-000-002        (23), -- Better then  0.000,002 Seconds
   time-000-000-001        (24), -- Better then  0.000,001 Seconds
                                 -- Better then one micro second
   time-000-000-000-5      (25), -- Better then  0.000,000,5 Seconds
   time-000-000-000-2      (26), -- Better then  0.000,000,2 Seconds
   time-000-000-000-1      (27), -- Better then  0.000,000,1 Seconds
   time-000-000-000-05     (28), -- Better then  0.000,000,05 Seconds
   time-000-000-000-02     (29), -- Better then  0.000,000,02 Seconds
   time-000-000-000-01     (30), -- Better then  0.000,000,01 Seconds
   time-000-000-000-005    (31), -- Better then  0.000,000,005 Seconds
   time-000-000-000-002    (32), -- Better then  0.000,000,002 Seconds
   time-000-000-000-001    (33), -- Better then  0.000,000,001 Seconds
                                 -- Better then one nano second
   time-000-000-000-000-5  (34), -- Better then  0.000,000,000,5 Seconds
   time-000-000-000-000-2  (35), -- Better then  0.000,000,000,2 Seconds
   time-000-000-000-000-1  (36), -- Better then  0.000,000,000,1 Seconds
   time-000-000-000-000-05 (37), -- Better then  0.000,000,000,05 Seconds
   time-000-000-000-000-02 (38), -- Better then  0.000,000,000,02 Seconds
   time-000-000-000-000-01 (39)  -- Better then  0.000,000,000,01 Seconds
   }


-- DE_TimeMark (Desc Name) Record 115
TimeMark ::= INTEGER (0..12002)
```

```
   -- In units of 1/10th second from local UTC time
   -- A range of 0~600 for even minutes, 601~1200 for odd minutes
   -- 12001 to indicate indefinite time
   -- 12002 to be used when value undefined or unknown


-- DE_J1939-71-Tire Leakage Rate (Desc Name) Record 116
TireLeakageRate ::= INTEGER (0..65535)


-- DE_J1939-71-Tire Location (Desc Name) Record 117
TireLocation ::= INTEGER (0..255)


-- DE_J1939-71-Tire Pressure (Desc Name) Record 118
TirePressure ::= INTEGER (0..1000)


-- DE_J1939-71-Tire Pressure Threshold Detection (Desc Name) Record 119
TirePressureThresholdDetection ::= ENUMERATED {
   noData               (0),  -- B'000'
   overPressure         (1),  -- B'001'
   noWarningPressure    (2),  -- B'010'
   underPressure        (3),  -- B'011'
   extremeUnderPressure (4),  -- B'100'
   undefined            (5),  -- B'101'
   errorIndicator       (6),  -- B'110'
   notAvailable         (7),  -- B'111'
   ... -- # LOCAL_CONTENT
   }


-- DE_J1939-71-Tire Temp (Desc Name) Record 120
TireTemp ::= INTEGER (0..65535)


-- DE_TractionControlState (Desc Name) Record 121
TractionControlState ::= ENUMERATED {
   unavailable (0), -- B'00  Not Equipped with tracton control
                    -- or tracton control status is unavailable
   off         (1), -- B'01  tracton control is Off
   on          (2), -- B'10  tracton control is On (but not Engaged)
   engaged     (3)  -- B'11  tracton control is Engaged
   }
   -- Encoded as a 2 bit value


-- DE_J1939-71-Trailer Weight (Desc Name) Record 122
TrailerWeight ::= INTEGER (0..65535)


-- DE_TransitPreEmptionRequest (Desc Name) Record 123
TransitPreEmptionRequest ::= ENUMERATED {
     typeOne       (0),
     typeTwo       (1),
     typeThree     (2),
     typeFour      (3),
     ... -- # LOCAL_CONTENT
     }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use


-- DE_TransitStatus (Desc Name) Record 124
TransitStatus ::= BIT STRING {
    none        (0), -- nothing is active
    anADAuse    (1), -- an ADA access is in progress (wheelchairs, kneling, etc.)
    aBikeLoad   (2), -- loading of a bicyle is in progress
    doorOpen    (3), -- a vehicle door is open for passenger access
    occM        (4),
    occL        (5)
    -- bits four and five are used to relate the
    -- the relative occupancy of the vehicle, with
    -- 00  as least full and 11 indicating a
```

```
    -- close-to or full conditon
    } (SIZE(6))


-- DE_TransmissionState (Desc Name) Record 125
TransmissionState ::= ENUMERATED {
   neutral         (0), -- Neutral, speed relative to the vehicle alignment
   park            (1), -- Park, speed relative the to vehicle alignment
   forwardGears    (2), -- Forward gears, speed relative the to vehicle alignment
   reverseGears    (3), -- Reverse gears, speed relative the to vehicle alignment
   reserved1       (4),
   reserved2       (5),
   reserved3       (6),
   unavailable     (7), -- not-equipped or unavailable value,
                        -- speed relative to the vehicle alignment

   ... -- # LOCAL_CONTENT
   }


-- DE_TravelerInfoType (Desc Name) Record 126
TravelerInfoType ::= ENUMERATED {
     unknown            (0),
     advisory           (1),
     roadSignage        (2),
     commercialSignage  (3),
     ... -- # LOCAL_CONTENT
     }
     -- values to 127 reserved for std use
     -- values 128 to 255 reserved for local use


-- DE_TransmitInterval (Desc Name) Record 127
TxTime ::= INTEGER (1..20)   -- units of seconds


-- DE_UniqueMSG_ID (Desc Name) Record 128
UniqueMSGID ::= OCTET STRING (SIZE(9))


-- DE_URL_Base (Desc Name) Record 129
URL-Base ::= IA5String (SIZE(1..45))


-- DE_URL_Link (Desc Name) Record 130
URL-Link ::= IA5String (SIZE(1..255))


-- DE_URL_Short (Desc Name) Record 131
URL-Short ::= IA5String (SIZE(1..15))


-- DE_VehicleHeight (Desc Name) Record 132
VehicleHeight ::= INTEGER (0..127)
    -- the height of the vehicle
    -- LSB units of 5 cm, range to 6.35 meters


-- DE_VehicleLaneAttributes (Desc Name) Record 133
VehicleLaneAttributes ::= INTEGER (0..65535)
-- With bits as defined:
    noLaneData              VehicleLaneAttributes ::= 0
                            -- ('0000000000000000'B)
    egressPath              VehicleLaneAttributes ::= 1
                            -- ('0000000000000001'B)
                            -- a two-way path or an outbound
                            -- path is described
    maneuverStraightAllowed VehicleLaneAttributes ::= 2
                            -- ('0000000000000010'B)
    maneuverLeftAllowed     VehicleLaneAttributes ::= 4
                            -- ('0000000000000100'B)
    maneuverRightAllowed    VehicleLaneAttributes ::= 8
                            -- ('0000000000001000'B)
    yield                   VehicleLaneAttributes ::= 16
```

```
                            -- ('0000000000010000'B)
    maneuverNoUTurn         VehicleLaneAttributes ::= 32
                            -- ('0000000000100000'B)
    maneuverNoTurnOnRed     VehicleLaneAttributes ::= 64
                            -- ('0000000001000000'B)
    maneuverNoStop          VehicleLaneAttributes ::= 128
                            -- ('0000000010000000'B)
    noStop                  VehicleLaneAttributes ::= 256
                            -- ('0000000100000000'B)
    noTurnOnRed             VehicleLaneAttributes ::= 512
                            -- ('0000001000000000'B)
    hovLane                 VehicleLaneAttributes ::= 1024
                            -- ('0000010000000000'B)
    busOnly                 VehicleLaneAttributes ::= 2048
                            -- ('0000100000000000'B)
    busAndTaxiOnly          VehicleLaneAttributes ::= 4096
                            -- ('0001000000000000'B)
    maneuverHOVLane         VehicleLaneAttributes ::= 8192
                            -- ('0010000000000000'B)
    maneuverSharedLane      VehicleLaneAttributes ::= 16384
                            -- ('0100000000000000'B)
                            -- a "TWLTL" (two way left turn lane)
    maneuverBikeLane        VehicleLaneAttributes ::= 32768
                            -- ('1000000000000000'B)


-- DE_VehicleLength (Desc Name) Record 134
VehicleLength ::= INTEGER (0..16383) -- LSB units are 1 cm


-- DE_VehicleMass (Desc Name) Record 135
VehicleMass ::= INTEGER (1..127) -- mass with an LSB of 50 Kg


-- DE_VehicleRequestStatus (Desc Name) Record 136
VehicleRequestStatus ::= OCTET STRING (SIZE(1))
  -- With bits set as follows:
  -- Bit 7 (MSB)  Brakes-on, see notes for use
  -- Bit 6 Emergency Use or operation
  -- Bit 5 Lights in use (see also the light bar element)
  -- Bits 5~0
  -- when a priority, map the values of
  -- LightbarInUse to the lower 4 bits
  -- and set the 5th bit to zero
  -- when a preemption, map the values of
  -- TransistStatus to the lower 5 bits


-- DE_VehicleStatusDeviceTypeTag (Desc Name) Record 137
VehicleStatusDeviceTypeTag ::= ENUMERATED {
   unknown             (0),
   lights              (1),  -- Exterior Lights
   wipers              (2),  -- Wipers
   brakes              (3),  -- Brake Applied
   stab                (4),  -- Stability Control
   trac                (5),  -- Traction Control
   abs                 (6),  -- Anti-Lock Brakes
   sunS                (7),  -- Sun Sensor
   rainS               (8),  -- Rain Sensor
   airTemp             (9),  -- Air Temperature
   steering            (10),
   vertAccelThres      (11), -- Wheel that Exceeded the
   vertAccel           (12), -- Vertical g Force Value
   hozAccelLong        (13), -- Longitudinal Acceleration
   hozAccelLat         (14), -- Lateral Acceleration
   hozAccelCon         (15), -- Acceleration Confidence
   accel4way           (16),
   confidenceSet       (17),
   obDist              (18), -- Obstacle Distance
   obDirect            (19), -- Obstacle Direction
   yaw                 (20), -- Yaw Rate
   yawRateCon          (21), -- Yaw Rate Confidence
   dateTime            (22), -- complete time
   fullPos             (23), -- complete set of time and
                             -- position, speed, heading
   position2D          (24), -- lat, long
   position3D          (25), -- lat, long, elevation
   vehicle             (26), -- height, mass, type
   speedHeadC          (27),
```

```
   speedC             (28),

   ... -- # LOCAL_CONTENT
   }
   -- values to 127 reserved for std use
   -- values 128 to 255 reserved for local use


-- DE_VehicleType (Desc Name) Record 138
VehicleType ::= ENUMERATED {
   none                 (0),  -- Not Equipped, Not known or unavailable
   unknown              (1),  -- Does not fit any other category
   special              (2),  -- Special use
   moto                 (3),  -- Motorcycle
   car                  (4),  -- Passenger car
   carOther             (5),  -- Four tire single units
   bus                  (6),  -- Buses
   axleCnt2             (7),  -- Two axle, six tire single units
   axleCnt3             (8),  -- Three axle, single units
   axleCnt4             (9),  -- Four or more axle, single unit
   axleCnt4Trailer      (10), -- Four or less axle, single trailer
   axleCnt5Trailer      (11), -- Five or less axle, single trailer
   axleCnt6Trailer      (12), -- Six or more axle, single trailer
   axleCnt5MultiTrailer (13), -- Five or less axle, multi-trailer
   axleCnt6MultiTrailer (14), -- Six axle, multi-trailer
   axleCnt7MultiTrailer (15),  -- Seven or more axle, multi-trailer
   ... -- # LOCAL_CONTENT
   }
   -- values to 127 reserved for std use
   -- values 128 to 255 reserved for local use


-- DE_VehicleWidth (Desc Name) Record 139
VehicleWidth ::= INTEGER (0..1023) -- LSB units are 1 cm


-- DE_VerticalAcceleration (Desc Name) Record 140
VerticalAcceleration ::= INTEGER (-127..127)
   -- LSB units of 0.02 G steps over
   -- a range +1.54 to -3.4G
   -- and offset by 50  Value 50 = 0g, Value 0 = -1G
   -- value +127 = 1.54G,
   -- value -120 = -3.4G
   -- value -121 for ranges -3.4 to -4.4G
   -- value -122 for ranges -4.4 to -5.4G
   -- value -123 for ranges -5.4 to -6.4G
   -- value -124 for ranges -6.4 to -7.4G
   -- value -125 for ranges -7.4 to -8.4G
   -- value -126 for ranges larger than -8.4G
   -- value -127 for unavailable data


-- DE_VerticalAccelerationThreshold (Desc Name) Record 141
VerticalAccelerationThreshold ::= BIT STRING {
   allOff      (0), -- B'0000  The condition All Off or not equipped
   leftFront   (1), -- B'0001  Left Front Event
   leftRear    (2), -- B'0010  Left Rear Event
   rightFront  (4), -- B'0100  Right Front Event
   rightRear   (8)  -- B'1000  Right Rear Event
   } -- to fit in 4 bits


-- DE_VINstring, (Desc Name) Record 142
VINstring ::= OCTET STRING (SIZE(1..17))
   -- A legal VIN or a shorter value
   -- to provide an ident of the vehicle
   -- If a VIN is sent, then IA5 encoding
   -- shall be used


-- DE_J1939-71-Wheel End Elect. Fault (Desc Name) Record 143
WheelEndElectFault ::= BIT STRING {
   bitOne      (1),
   bitTwo      (2),
   bitThree    (3),
   bitFour     (4)
   }
```

```
-- DE_J1939-71-Wheel Sensor Status (Desc Name) Record 144
WheelSensorStatus ::= ENUMERATED {
     off           (0),
     on            (1),
     notDefined    (2),
     notSupoprted  (3)
     }



-- DE_WiperRate (Desc Name) Record 145
WiperRate ::= INTEGER (0..127) -- units of sweeps per minute



-- DE_WiperStatusFront (Desc Name) Record 146
WiperStatusFront ::= ENUMERATED {
     unavailable         (0), -- Not Equipped with wiper status
                              -- or wiper status is unavailable
     off                 (1),
     intermittent        (2),
     low                 (3),
     high                (4),
     washerInUse       (126), -- washing solution being used
     automaticPresent  (127), -- Auto wiper equipped
     ... -- # LOCAL_CONTENT
     }



-- DE_WiperStatusRear (Desc Name) Record 147
WiperStatusRear ::= ENUMERATED {
     unavailable         (0), -- Not Equipped with wiper status
                              -- or wiper status is unavailable
     off                 (1),
     intermittent        (2),
     low                 (3),
     high                (4),
     washerInUse       (126), -- washing solution being used
     automaticPresent  (127), -- Auto wipper equipped
     ... -- # LOCAL_CONTENT
     }



-- DE_YawRate (Desc Name) Record 148
YawRate ::= INTEGER (-32767..32767)
   -- LSB units of 0.01 degrees per second (signed)



-- DE_YawRateConfidence (Desc Name) Record 149
YawRateConfidence ::= ENUMERATED {
   unavailable    (0), -- B'000  Not Equipped with yaw rate status
                       -- or yaw rate status is unavailable
   degSec-100-00  (1), -- B'001  100  deg/sec
   degSec-010-00  (2), -- B'010  10   deg/sec
   degSec-005-00  (3), -- B'011  5    deg/sec
   degSec-001-00  (4), -- B'100  1    deg/sec
   degSec-000-10  (5), -- B'101  0.1  deg/sec
   degSec-000-05  (6), -- B'110  0.05 deg/sec
   degSec-000-01  (7)  -- B'111  0.01 deg/sec
   }
   -- Encoded as a 3 bit value



END
-- end of the DSRC module.



-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--
-- Start of External Data entries...
-- Grouped into sets of modules
-- -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
--
-- ^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-
--
```

```
-- Begin module: NTCIP
--
-- ^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^
NTCIP DEFINITIONS AUTOMATIC TAGS::= BEGIN



-- ESS_EssMobileFriction (Desc Name) Record 1
-- From source: NTCIP 1204
EssMobileFriction ::= INTEGER (0..101)

-- ESS_EssPrecipRate_quantity (Desc Name) Record 2
-- From source: NTCIP 1204
EssPrecipRate ::= INTEGER (0..65535)

-- ESS_EssPrecipSituation_code (Desc Name) Record 3
-- From source: NTCIP 1204
EssPrecipSituation ::= ENUMERATED {
   other (1),
   unknown (2),
   noPrecipitation (3),
   unidentifiedSlight (4),
   unidentifiedModerate (5),
   unidentifiedHeavy (6),
   snowSlight (7),
   snowModerate (8),
   snowHeavy (9),
   rainSlight (10),
   rainModerate (11),
   rainHeavy (12),
   frozenPrecipitationSlight (13),
   frozenPrecipitationModerate (14),
   frozenPrecipitationHeavy (15)
   }

-- ESS_EssPrecipYesNo_code (Desc Name) Record 4
-- From source: NTCIP 1204
EssPrecipYesNo ::= ENUMERATED {precip (1), noPrecip (2), error (3)}

-- ESS_EssSolarRadiation_quantity (Desc Name) Record 5
-- From source: NTCIP 1204
EssSolarRadiation ::= INTEGER (0..65535)


-- Inserting file: NTCIPstubs.txt here.


-- This is a collection of code and stubs needed to match to the
-- NTCIP (and ESS) work.


-- End of inserted file


END
-- End of the NTCIP module.

-- ^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^
--
-- Begin module: ITIS
--
-- ^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^
ITIS DEFINITIONS AUTOMATIC TAGS::= BEGIN



-- DE_Incident Response Equipment (Desc Name) Record 6
-- From source: SAE ITIS Terms
IncidentResponseEquipment ::= ENUMERATED {
   ground-fire-suppression          (9985),
   heavy-ground-equipment           (9986),
   aircraft                         (9988),
   marine-equipment                 (9989),
   support-equipment                (9990),
   medical-rescue-unit              (9991),
   other                            (9993),   -- Depreciated by fire standards, do not
                                              -- use
   ground-fire-suppression-other    (9994),
   engine                           (9995),
   truck-or-aerial                  (9996),
   quint                            (9997),   -- A five-function type of fire apparatus.
                                              -- The units in the movie Backdraft were
                                              -- quints
```

```
    tanker-pumper-combination        (9998),
    brush-truck                      (10000),
    aircraft-rescue-firefighting     (10001),
    heavy-ground-equipment-other     (10004),
    dozer-or-plow                    (10005),
    tractor                          (10006),
    tanker-or-tender                 (10008),
    aircraft-other                   (10024),
    aircraft-fixed-wing-tanker       (10025),
    helitanker                       (10026),
    helicopter                       (10027),
    marine-equipment-other           (10034),
    fire-boat-with-pump              (10035),
    boat-no-pump                     (10036),
    support-apparatus-other          (10044),
    breathing-apparatus-support      (10045),
    light-and-air-unit               (10046),
    medical-rescue-unit-other        (10054),
    rescue-unit                      (10055),
    urban-search-rescue-unit         (10056),
    high-angle-rescue                (10057),
    crash-fire-rescue                (10058),
    bLS-unit                         (10059),
    aLS-unit                         (10060),
    mobile-command-post              (10075),  -- Depreciated, do not use
    chief-officer-car                (10076),
    hAZMAT-unit                      (10077),
    type-i-hand-crew                 (10078),
    type-ii-hand-crew                (10079),
    privately-owned-vehicle          (10083),  -- (Often found in volunteer fire teams)
    other-apparatus-resource         (10084),  -- (Remapped from fire code zero)
    ambulance                        (10085),
    bomb-squad-van                   (10086),
    combine-harvester                (10087),
    construction-vehicle             (10088),
    farm-tractor                     (10089),
    grass-cutting-machines           (10090),
    hAZMAT-containment-tow           (10091),
    heavy-tow                        (10092),
    light-tow                        (10094),
    flatbed-tow                      (10114),
    hedge-cutting-machines           (10093),
    mobile-crane                     (10095),
    refuse-collection-vehicle        (10096),
    resurfacing-vehicle              (10097),
    road-sweeper                     (10098),
    roadside-litter-collection-crews (10099),
    salvage-vehicle                  (10100),
    sand-truck                       (10101),
    snowplow                         (10102),
    steam-roller                     (10103),
    swat-team-van                    (10104),
    track-laying-vehicle             (10105),
    unknown-vehicle                  (10106),
    white-lining-vehicle             (10107),  -- Consider using Roadwork "road marking
                                               -- operations" unless the objective is to
                                               -- refer to the specific vehicle of this
                                               -- type.  Alternative Rendering: line
                                               -- painting vehicle
    dump-truck                       (10108),
    supervisor-vehicle               (10109),
    snow-blower                      (10110),
    rotary-snow-blower               (10111),
    road-grader                      (10112),  -- Alternative term: motor grader
    steam-truck                      (10113),  -- A special truck that thaws culverts and
                                               -- storm drains
    ... -- # LOCAL_CONTENT_ITIS
    }

-- EXT_ITIS_Codes (Desc Name) Record 7
-- From source: Adopted SAE J2540-2  (ITIS Phrases), March 2002
ITIScodes ::= INTEGER (0..65565)
-- The defined list of ITIS codes is too long to list here
-- Many smaller lists use a sub-set of these codes as defined elements
-- Also enumerated values expressed as text constant are very common,
-- and in many deployments the list codes are used as a shorthand for
-- this text.  Also the XML expressions commonly use a union of the
-- code values and the textual expressions.
-- Consult SAE J2540 for further details.

-- DF_ITIS-Codes_And_Text (Desc Name) Record 8
-- From source: Adopted SAE J2540-2  (ITIS Phrases), March 2002
ITIScodesAndText ::=  SEQUENCE (SIZE(1..100)) OF SEQUENCE {
```

```
  item CHOICE    {
       itis ITIScodes,
       text ITIStext
       } -- # UNTAGGED
  }

-- DE_ITIS_Text (Desc Name) Record 9
-- From source: Adopted SAE J2540-2  (ITIS Phrases), March 2002
ITIStext ::= IA5String (SIZE(1..500))

-- DE_Responder Group Affected (Desc Name) Record 10
-- From source: SAE ITIS Terms
ResponderGroupAffected ::= ENUMERATED  {
   emergency-vehicle-units           (9729),  -- Default phrase, to be used when one of
                                              -- the below does not fit better
   federal-law-enforcement-units     (9730),
   state-police-units                (9731),
   county-police-units               (9732),  -- Hint: also sheriff response units
   local-police-units                (9733),
   ambulance-units                   (9734),
   rescue-units                      (9735),
   fire-units                        (9736),
   hAZMAT-units                      (9737),
   light-tow-unit                    (9738),
   heavy-tow-unit                    (9739),
   freeway-service-patrols           (9740),
   transportation-response-units     (9741),
   private-contractor-response-units (9742),
   ... -- # LOCAL_CONTENT_ITIS
   }
   -- These groups are used in coordinated response and staging area information
   -- (rather than typically consumer related)


-- DE_Vehicle Groups Affected (Desc Name) Record 11
-- From source: SAE ITIS Terms
VehicleGroupAffected ::= ENUMERATED  {
   all-vehicles                                 (9217),
   bicycles                                     (9218),
   motorcycles                                  (9219),  -- to include mopeds as well
   cars                                         (9220),  -- (remapped from ERM value of
                                                         -- zero)
   light-vehicles                               (9221),
   cars-and-light-vehicles                      (9222),
   cars-with-trailers                           (9223),
   cars-with-recreational-trailers              (9224),
   vehicles-with-trailers                       (9225),
   heavy-vehicles                               (9226),
   trucks                                       (9227),
   buses                                        (9228),
   articulated-buses                            (9229),
   school-buses                                 (9230),
   vehicles-with-semi-trailers                  (9231),
   vehicles-with-double-trailers                (9232),  -- Alternative Rendering:  western
                                                         -- doubles
   high-profile-vehicles                        (9233),
   wide-vehicles                                (9234),
   long-vehicles                                (9235),
   hazardous-loads                              (9236),
   exceptional-loads                            (9237),
   abnormal-loads                               (9238),
   convoys                                      (9239),
   maintenance-vehicles                         (9240),
   delivery-vehicles                            (9241),
   vehicles-with-even-numbered-license-plates   (9242),
   vehicles-with-odd-numbered-license-plates    (9243),
   vehicles-with-parking-permits                (9244),
   vehicles-with-catalytic-converters           (9245),
   vehicles-without-catalytic-converters        (9246),
   gas-powered-vehicles                         (9247),
   diesel-powered-vehicles                      (9248),
   lPG-vehicles                                 (9249),
   military-convoys                             (9250),
   military-vehicles                            (9251),
   ... -- # LOCAL_CONTENT_ITIS
   }
   -- Classification of vehicles and types of transport


END
-- End of the ITIS module.
```

```
-- End of file output at 11/11/2009 1:15:00 PM
```