SAE International™

SURFACE VEHICLE STANDARD

SAE J2630

ISSUED DEC2003

Issued 2003-12

Converting ATIS Message Standards From ASN.1 to XML

Foreword—This document provides a set of conversion rules and recommendations which can be applied to ASN.1 to produce XML schema outputs. These rules were developed primarily for use with the ITS message sets developed by SAE and other standards organizations. However, these rules may also apply to other ITS message sets developed by SAE and other standards organizations. The results of applying these rules (a set of XML schema encodings) is published elsewhere (typically with the source standard). In the case of SAE standards the results of applying these rules can be found in the revised edition of SAE J2354.

TABLE OF CONTENTS

1. ***Scope***—This SAE Standard presents a set of rules for transforming an Abstract Syntax Notation (ASN.1) message set definition into an eXtensible Markup Language (XML) schema. The result is intended to be a stand-alone XML Schema that is fully consistent with an existing ASN.1 information model. This is a different goal from other related work by other standards bodies developing a set of XML encoding rules for ASN.1 or ASN.1 encodings for XML Schema. These rules were initially developed in order to produce an XML schema for the SAE ATIS standard. While other standards may also choose to use these rules, the rules may not be applicable for all environments.

The goal for these transformation rules is twofold. The first goal is to provide a uniform set of such rules that all interested parties can use. The second goal is to use such rules to define an adopted schema for traveler information that reflects the preferred translation of ASN.1 message sets to XML for use by ITS system implementers. The first goal is met by this document. The second goal is met by employing this document to produce XML information as part of the periodic re-balloting of the SAE ATIS standard. This is a parallel standards effort with this document.[1]

1. At this time a numer o technical data formatting issues are being resolved as part of the re-issue of SAE J2354 (drafts available at the committee forum site t http://www.SAE.org/). The XML productions will be included in this parallel effort. SAE J2353 and the message set contents of other ITS standards are being integrated within this single standard to produce a composite data dictionary of all SAE ITS messages.

These rules were developed as part of the process to draft an XML version of the ATIS data element and message set standards. The original effort focused upon the needs found in the currently adopted October 2000 SAE J2354 standard for ATIS message sets. SAE J2354 makes extensive use of elements from the ITETMDD work and from ITE-TCIP work. In addition, the draft “Event Report Message” (ERM) portion of the Message Sets for External Traffic Management Center Communication (MS/ETMCC) standard was also examined and translated. Message and data elements from the IEEE Incident Management standard (IEEE 1512-2000) were also examined. By this effort, every major message set of ITS was considered to some degree to ensure that the resulting translations could be successfully employed by others toward a common result. It is intended that the resulting ATIS schema will be voted on as an SAE ATIS standard. Other standards-developing organizations may also choose to use these rules; however, it is recognized that translations may not be necessary in some environments and that these rules may not be applicable to all environments within ITS. It is left to each standards-development organization to make such determinations for their specific environments.

Section 4 is the complete set of conversion rules. An example of use for each rule is given, showing an original ASN.1 definition, the resulting XML schema definition, and a sample XML document element where applicable.

Section 5 of this document discusses the background of some non-obvious conversion rules presented in Section 4.[2] These include methods of handling the XML representation of enumerations, octet strings, and bit strings. Section 5 also describes how XML namespaces can be used to point to types defined in the Traffic Management Data Dictionary (TMDD), the Transit Communications Interface Profiles (TCIP) standard, the Location Reference Message Set (LRMS), or the International Traveler Information Systems (ITIS) phrase lists. It also discusses additional translation refinements proposed for the conversion of the ATIS message standard.

Some omissions in the SAE standards required new types to be defined in order to have a valid XML schema. In addition, some new types must be defined to implement the representation of octet strings and bit strings. These changes are documented in Section 6. In the currently adopted revision of the standard the use of inline definitions rather than the more proper use of formal Type Definitions is also a challenge for conversion.

Section 6 also documents a number of common types that were defined globally at the start of the schema rather than being redefined identically, in-line multiple times in the schema.

Section 7 presents a sizable sample ASN.1 definition and the corresponding XML schema. The sample illustrates a large number of the conversion rules.

2. References

2.1 Applicable Publications—The following publications form a part of this specification to the extent specified herein.

2.1.1 SAE PUBLICATIONS—Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001.

SAE J2353—Data Dictionary for Advanced Traveler Information Systems (ATIS)
SAE J2354—Message Sets for Advanced Traveler Information Systems (ATIS)

Readers are advised that these two documents are now being updated to reflect current data registry formats and current ITS industry practices. J2354 will also include an XML schema derived from the ASN.1 found in the revised documents meeting the translation rules of this standard.

2. This document deals only with the final set of recommended rules. An earlier set of preliminary rules, refined by group experience, is documented in several informal reports. This earlier draft is available at the committee forum site at http://www.SAE.org.

2.1.2 ITE PUBLICATIONS—Available from Institute for Transportation Engineers (ITE), 1099 14th Street, NW, Suite 300 West, Washington, DC 20005-3438 Telephone: 202-289-0222 ite_staff@ite.org Transit Communications Interface Profiles (TCIP) Framework Standard (NTCIP 1400 Draft), December 2000.

Transit Communications Interface Profiles (TCIP) Scheduling Business Area Standard Framework Standard (NTCIP 1403 Draft), December 2000.
Transit Communications Interface Profiles (TCIP) Passenger Information Business Area Standard Framework Standard (NTCIP 1403 Draft), December 2000.
Standards for Functional Level Traffic Management Data Dictionary (TMDD), Standard No. TM1.01.4.
Message Sets for External Traffic Management Center Communication (MS/ETMCC) ITE-AASHTO-TM2.01 August 2000.

2.1.3 PUBLICATION—XML Definitions Available on the Internet from the URLs below.

World-Wide-Web Consortium Definition of XML: http://www.w3.org/XML/
World-Wide-Web Consortium Definition of XML Schemas: http://www.w3.org/XML/Schema Consisting of Parts Two and Three. [Part One is informative and is listed in the Bibliography]

3. Definitions

3.1 Application Program—In the context of this document, any process beyond the XML validation, which may or may not be done by the top most application. For example it is common to employ intermediate levels to further process the XML source and manipulate it based on knowledge of the message structures or <xsd:appinfo> tag data.

3.2 ATIS—Advanced Traveler Information Systems, specifically the SAE J2353 and J2354 standards of ITS data elements and message sets. As a slang term, this has become the common term to refer to all the SAE efforts in developing ITS messages.

3.3 ERM—Event Reporting Message. A portion of the TMDD message set family which was develop first by rural states and has now been added to the most current draft of the ITE committee message set work.

3.4 False—In the context of this standard and XML, the Boolean value 0 is also defined as False.

3.5 IEEE—The Institute of Electrical and Electronics Engineers, a New York not-for-profit corporation engaged in developing standards. One of the standards developing organizations. A party to the cooperative development agreement of the FHWA to develop ITS standards.

3.6 IM—Incident Management, a slang term for the IEEE 1512-2000 and related family of standards that deal with exchanges of message sets between centers involved in an incident.

3.7 ITE—The Institute of Traffic Engineers, a not-for-profit corporation engaged in developing standards. One of the standards developing organizations. A party to the cooperative development agreement of the FHWA to develop ITS standards.

3.8 LRMS—Location Referencing Message Set, a portion of ITS message structures developed under the FHWA cooperative agreements and now undergoing standardization by the SAE in its second major revision. The LRMS handles all of the spatial portions of ITS messages and is therefore widely re-used by other standards.

3.9 TCIP—Transit Communications Interface Profiles

3.10 TMDD—Traffic Management Data Dictionary. The data dictionary developed by the ITE as part of its work on messages used between and within traffic management centers. Also often used as a slang term for the both the data elements and the messages developed in that effort. The SAE ATIS message set reuses these data elements as well.

3.11 True—In the context of this standard and XML, the Boolean value 1 is also defined as True.

3.12 XML—eXtensible Marked Language

3.13 XSL—eXtensible Stylesheet Language

4. Rules for Converting an ASN.1 Message Set Definition to an XML Schema—The following is a list of rules organized as pseudo-code instructions for converting an ASN.1 definition to an XML schema. It is far from complete for anything that may appear in ASN.1, but it is believed complete for everything that appeared in the ATIS message sets standard as well as several other standards found in ITS.

There are many ways such a conversion could be performed; this approach attempts to be as direct as possible. For example, almost no user-defined attributes are used, and all data structures are preserved. The information contained in the transmitted document is identical, and the data structures are clearly equivalent.

One goal is that a program with knowledge of an ASN.1 definition set and the XML schema produced from it by these rules should be able to convert a conforming XML document into a conforming ASN.1 encoding and vice versa.[3] A second goal, however, is that developers of XML-based interfaces and the software they develop will not require knowledge or use of ASN.1.

Note: In ASN.1, formally defined type definitions begin with a capital letter and identifiers for those types begin with a lower case letter. This conversion to XML preserves this convention, although there is no rule in XML requiring this convention. In fact, ebXML guidelines recommend that the names of all types and elements should begin with a capital letter, while the names of attributes should begin with a lower case letter. The SAE ATIS committee should consider whether to retain the lower case starting letter convention. In any event, shortening of some ASN.1 names when used as tags may be adopted.

The examples are taken from fragments of SAE ATIS, ITE TMDD, and IEEE IM standards, but the precise structures have been modified at times to serve as an example more clearly. Explanatory comments from the ASN.1 have been removed to conserve space. Consult the governing standard for correct structures and use commentary. The indentation shown in this document is purely for readability; no recommendation in this regard is given and any style desired may be adopted. Note that the convention is maintained that all defined type names start with an upper-case letter and all defined element names start with a lower-case letter.

In general, XML schemas define types and elements. Elements are instances of types. Basic types are predefined, such as integer, decimal, date, or string. User-defined types are defined as restrictions, expansions, or unions of other types. A simple type does not contain any internal structure. A complex type is a structure containing other elements or structures, defined as a SEQUENCE or SET. It is expected that the order of elements found in a complex type structure will be preserved (e.g., arbitrary re-ordering shall not be allowed).

Type names do not appear in XML documents; only the names of elements appear as tags in an XML document. Type definitions and global element definitions may be imported and exported among schemas using namespaces, but non-global element definitions may not.

3. There is some debate on the complete ability to convert back from XML to the original ASN.1. This is a laudable goal and any differences would be expected to be minor. There is a small number of XML expressions which have no equivalent in ASN.1 and hte definitive translations of these would be problematic to ASN.1 users.

Sections 4.1 through 4.7 present rules for converting ASN.1 simple-type definitions to the corresponding XML simple type definitions. Sections 4.8 through 4.12 present rules for converting ASN.1 complex type definitionsto the corresponding XML complex type definitions. Sections 4.13 through 4.18 present rules for converting ASN.1 element definitions to the corresponding XML element definitions. Sections 4.19 through 4.25 present other rules to follow in creating an XML schema from an ASN.1 definition.

4.1 Defining a Simple Type in Terms of a Basic Type—To define a new simple type in terms of an existing basic type, replace the ASN.1 definition *Abc ::= basictype*

```
<xsd:simpleType name="Abc">
   <xsd:restriction base="xsd:basictype"/>
</xsd:simpleType>
```

where basic type is chosen from the following table:

TABLE 1—XML TRANSLATION BASIC TYPES

ASN.1 basic type	XML basic type
Any date	xsd:date
Any date + time	xsd:dateTime
Any time	xsd:time
BOOLEAN	xsd:boolean
IA5STRING	xsd:string
INTEGER	xsd:integer -- See Section 4.2 for further rules
NumericString	xsd:string
PrintableString	xsd:string
REAL	xsd:decimal or xsd:float
UTF8String	xsd:string
VisibleString	xsd:string

The following types are not considered basic XML types and are handled in the sections that follow.

TABLE 2—XML TRANSLATION RECOMMENDATIONS

ASN.1 basic type	XML Recommendations
ASN.1 basic type	XML Recommendations
ASN.1 basic type	XML Recommendations
ASN.1 basic type	XML Recommendations
ASN.1 basic type	XML Recommendations
ASN.1 basic type	XML Recommendations

4.2 Defining Basic INTEGER Types—For definitions of integers, derived XML classes may be used if the defined values fall within the minimum and maximum (inclusive) values specified in the table below. XML basic integer types are defined in the following table.

TABLE 3—RANGES FOR XML INTEGER TYPES

Minimum value	Maximum value	XML type
–2147483648	2147483647	xsd:int
–32768	32767	xsd:short
–128	127	xsd:byte
0	4294967295	xsd:unsignedInt
0	65535	xsd:unsignedShort
0	255	xsd:unsignedByte

When no further information on the value range is available, the type xsd:integer shall be used.

For example, the ASN.1 type definitions:

```
DatabaseNumber ::= INTEGER (0..255)
```

The resulting schema fragment would be

```
<xsd:simpleType name="DatabaseNumber">
   <xsd:restriction base="xsd:unsignedByte"/>
</xsd:simpleType>
```

4.3 **Defining INTEGER Types with Minimum and Maximum Values**—For an ASN.1 type defined as an integer with a minimum and maximum value, replace `Abc ::= INTEGER (a .. b)` by:

```
<xsd:simpleType name="Abc">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="a"/>
      <xsd:maxInclusive value="b"/>
   </xsd:restriction>
</xsd:simpleType>
```

xsd:integer may be replaced by one of the derived integer types listed in Table. If the lower bound is zero and the basic type is an unsigned type, the lower bound statement may be omitted since zero is already the lower bound of the unsigned type.

For example, given the ASN.1 fragments of:

```
Scaling         ::= INTEGER (0..15)
Start-time      ::= INTEGER (0..4095)
Location-long   ::= INTEGER (-180000000..180000000)
```

The resulting schema fragments would be (applying this rule and the prior one for integer ranges):

```
<xsd:simpleType name="Scaling">
   <xsd:restriction base="xsd:unsignedByte">
      <xsd:maxInclusive value="15"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="Start-time">
   <xsd:restriction base="xsd:unsignedShort">
      <xsd:maxInclusive value="4095"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="Location-long">
   <xsd:restriction base="xsd:int">
      <xsd:minInclusive value="-180000000"/>
      <xsd:maxInclusive value="180000000"/>
   </xsd:restriction>
</xsd:simpleType>
```

4.4 **Defining String Types of Minimum and Maximum Length—**For a type defined as a character string (typically an IA5String) of minimum and maximum length, replace

```
Abc ::= IA5String       (SIZE(a .. b)) or
Abc ::= NumericString   (SIZE(a .. b)) or
Abc ::= PrintableString(SIZE(a .. b)) or
Abc ::= VisibleString   (SIZE(a .. b)) by:
```

```
<xsd:simpleType name="Abc">
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="a"/>
      <xsd:maxLength value="b"/>
   </xsd:restriction>
</xsd:simpleType>
```

For example, given the ASN.1 fragment of:

```
Traveler-LastName     ::= IA5String (SIZE(1..25))
Traveler-Email        ::= IA5String (SIZE(1..40))
Route-Description     ::= IA5String (SIZE(1..300))
```

The resulting schema fragment would be:

```
<xsd:simpleType name="Traveler-LastName">
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="25"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="Traveler-Email">
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="40"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="Route-Description">
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="300"/>
   </xsd:restriction>
</xsd:simpleType>
```

4.5 **Defining String Types of Fixed Length**—For a type defined as a string (typically an IA5String or other derived ASN.1 string type) of fixed length, replace `Abc ::= IA5String (SIZE(a))` by:

```
<xsd:simpleType name="Abc">
   <xsd:restriction base="xsd:string">
   <xsd:length value="a"/>
   </xsd:restriction>
</xsd:simpleType>
```

For example, given the ASN.1 fragment of:

```
Flight-DestinationAirport  ::= IA5String (SIZE(3))
ATIS-TimeOffset            ::= IA5String (SIZE(4))
```

The resulting schema fragment would be:

```
<xsd:simpleType name="Flight-DestinationAirport">
   <xsd:restriction base="xsd:string">
      <xsd:length value="3"/>
   </xsd:restriction>


</xsd:simpleType>
<xsd:simpleType name="ATIS-TimeOffset ">
   <xsd:restriction base="xsd:string">
      <xsd:length value="4"/>
   </xsd:restriction>
</xsd:simpleType>
```

4.6 Defining ENUMERATED Types—For a type defined as an ENUMERATED type, replace `Abc ::= ENUMERATED {. . . }` by:

```
<xsd:simpleType name="Abc">
   <xsd:union>
      <xsd:simpleType>
         <xsd:restriction base="xsd:unsignedInt">
            . . .
         </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value=". . ."/>
            <xsd:enumeration value=". . ."/>
         </xsd:restriction>
      </xsd:simpleType >
   </xsd:union>
   <xsd:annotation>
      <xsd:appinfo>
         . . .
      </xsd:appinfo>
   </xsd:annotation>
</xsd:simpleType>
```

The base type of the integer index is set as per prior rules for the integer range of the index list. The possible values of the integer index are either enumerated (if there are only a few permissible values) or specified with minInclusive and maxInclusive statements. The permissible text values are enumerated. The <u>union</u> of the integer index with the enumerated list of text strings permits an element of the defined type to contain either an index or one of the permissible text strings. The optional <appinfo> section inside <annotation>tags permits index values to be linked with text values by an application program. Section 5.1 discusses this construction in greater detail.

If the original intention of the enumeration was to be as a bit string, then the numerical indexes should be omitted and only the textual enumerations shall be used.[4]

4. The *Free-Flow-Rate* data element adopted in the original version of SAE J2369 would be an example of this type of definition of an enumeration with binary overtones. List items include lines formatted as: under–80–mph (13) , -- ‘1101 ‘ B.

For example, given the ASN.1 definition:

```
ATIS-SearchOperator ::= ENUMERATED {
     mustContain         (0),
     shouldContain       (1),
     shouldNotContain    (2),
     mustNotContain      (3),
     ...
```

The resulting schema would be:

```
<xsd:simpleType name="ATIS-SearchOperator">
<xsd:union>
   <xsd:simpleType>
      <xsd:restriction base="xsd:unsignedByte">
          <xsd:maxInclusive value="3"/>
        </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType>
        <xsd:restriction base="xsd:string">
           <xsd:enumeration value="mustContain"/>
           <xsd:enumeration value="shouldContain"/>
           <xsd:enumeration value="shouldNotContain"/>
           <xsd:enumeration value="mustNotContain"/>
        </xsd:restriction>
     </xsd:simpleType >
  </xsd:union>
  <xsd:annotation>
     <xsd:appinfo>
        mustContain (0)
        shouldContain (1)
        shouldNotContain (2)
        mustNotContain (3)
     </xsd:appinfo>
  </xsd:annotation>
</xsd:simpleType>
```

If atis-SearchOperator is defined to be an element of type ATIS-SearchOperator, then an example of content using this definition would be

```
<atis-SearchOperator>mustContain</atis-SearchOperator>
```

An equally valid expression with the same meaning would be

```
<atis-SearchOperator>0</atis-SearchOperator>
```

If one or more of the index values in a range is undefined, then the missing elements shall be added to create a continuous range. If large gaps in range are present, a union for each smaller continuous range may be used.

If one or more of the integer index values is defined as a “local value,” then the following construct should beadded to the union:

```
<xsd:simpleType>
   <xsd:restriction base="xsd:string"/>
      <xsd:pattern value = "\[.*\].*"/>
   </xsd:restriction>
</xsd:simpleType>
```

This permits a value such as [StateDOT]SpecialCode to appear as the value of the element. Such a definition (1) tells anyone receiving the message whose special code it is (StateDOT’s), (2) enables StateDOT to write code to look for its SpecialCode value and handle it accordingly, (3) enables other parties who don’t have StateDOT’s list of special codes to validate the document without needing to know what the special code means.

Currently there is no rule for what to do with index values labeled as “reserved for future use.” If new integer values and corresponding text values are added to the standard, the standard will be changed by expanding the range of index values permitted and adding the new text value to the enumerated list of permissible text values.

4.7 **Defining a Type Definition Embedded in an Element Definition**—Sometimes an ASN.1 definition contains an embedded type definition. In these cases, the ASN.1 should be restructured so that the type is defined globally rather than within the scope of the data element, such as in the case below.

```
Direction ::= BIT STRING (SIZE(1))
   one-Direction   Direction ::= '0'B
   both-Directions Direction ::= '1'B
```

In such a case the ASN.1 may be re-defined for the purpose of converting to XML as an in-line definition of an enumerated type, such as:

```
Direction ::= BIT STRING (SIZE(1)) {
   one-Direction    (0),
   both-Directions (1)
   }
```

Once converted, the resulting enumeration is translated using the other rules in this document; refer to 4.6.

4.8 **Defining a Complex Type as a SEQUENCE of Elements**—Replace a complex type definition `Abc ::= SEQUENCE { ... } or Abc ::= SET { … }` by:

```
<xsd:complexType name="Abc">
   <xsd:sequence>
      ...
   </xsd:sequence>
</xsd:complexType>
```

For example, the ASN.1 fragment

```
DirectoryRequest ::= SEQUENCE  {
    coreRequest DirectoryCoreRequest,
    start DateTimePair OPTIONAL,
    end DateTimePair OPTIONAL
    }
```

is converted to

```
<xsd:complexType name="DirectoryRequest">
   <xsd:sequence>
      <xsd:element name="coreRequest" type="DirectoryCoreRequest"/>
      <xsd:element name="start" type="xsd:dateTime" minOccurs="0"/>
      <xsd:element name="end" type="xsd:dateTime" minOccurs="0"/>
   </xsd:sequence>


</xsd:complexType>
```

Refer to Section 4.20 for rules about optional elements.

4.9 Defining a Complex Type as a CHOICE of Elements—Replace a complex type definition `Abc ::= CHOICE { ... }` by:

```
<xsd:complexType name="Abc">
   <xsd:choice>
      ...
   </xsd:choice>
</xsd:complexType>
```

For example, the ASN.1 fragment

```
LinkOrNodeAdditionalInformation ::= CHOICE {
    link LinkAdditionalInformation,
    node NodeAdditionalInformation
    }
```

is converted to:

```
<xsd:complexType name="LinkOrNodeAdditionalInformation">
   <xsd:choice>
      <xsd:element name="link" type="LinkAdditionalInformation"/>
      <xsd:element name="node" type="NodeAdditionalInformation"/>
   </xsd:choice>
</xsd:complexType>
```

4.10 Defining a Complex Type as a SEQUENCE OF Named Types—A type defined as a sequence of multiple occurrences of a named type translates to a sequence of multiple occurrences of elements of that named type. For these definitions, replace: `Abc ::= SEQUENCE OF Def` or `Abc [number] SEQUENCE OF Def` by:

```
<xsd:complexType name="Abc">
   <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="def" type="Def"/>
   </xsd:sequence>
</xsd:complexType>
```

For example, given the ASN.1 definition of:

```
TrafficInformation ::= SEQUENCE OF LinkOrNode
```

The resulting schema definition would be:

```
<xsd:complexType name="TrafficInformation">
   <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="linkOrNode" type="LinkOrNode"/>
   </xsd:sequence>
</xsd:complexType>
```

Note that the type name (beginning with a capital letter) has been used with a lower case initial to name an element of that type. This new element in the sequence is not required by ASN.1 but is required in XML.

An example of content using this would be

```
<TrafficInformation>
   <linkOrNode>  complex contents of type LinkOrNode here </linkOrNode>
   <linkOrNode>  complex contents of type LinkOrNode here </linkOrNode>
   <linkOrNode>  complex contents of type LinkOrNode here </linkOrNode>
</TrafficInformation>
```

4.11 Defining a Complex Type as a SEQUENCE OF a Named Type with Minimum and Maximum Number of Elements—For a type defined as a sequence of a named type with a minimum and/or maximum number of elements, replace:

`Abc ::= SEQUENCE (a .. b)` of `Def` or `Abc [number] SEQUENCE (`*`a .. b`*`)` of `Def`

```
<xsd:complexType name="Abc">
   <xsd:sequence minOccurs = "a" maxOccurs="b">
      <xsd:element name="def" type="Def"/>
   </xsd:sequence>
</xsd:complexType>
```

For example, given the ASN.1 definition (taken from a message in the IEEE Incident Management Message set, 1512.3) of:

```
hazardClass SEQUENCE (SIZE (0..7)) OF HazardClass,
```

The resulting schema would be:

```
<xsd:complexType name="hazardClass">
        <xsd:sequence minOccurs = "0" maxOccurs = "7">
                <xsd:element name="hazardClass-item" type="HazardClass"/
        </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

Note that hazardClass could not be used as the name of the elements in the sequence, because that is the name of the sequence itself. Instead, the naming rule presented in the following section was employed.

4.12 Defining a Complex Type as a SEQUENCE OF an Unnamed Type—If a type is defined as a SEQUENCE OF a basic type (modified or not), then the rules of 4.10 and 4.11 cannot be used because there is no type name to be used as an element name. Two options have been proposed for dealing with this situation, as explained below.

Add "-item" to the name of the parent element to form the name of each element in the sequence. For example, replace `Abc ::= SEQUENCE OF INTEGER` by:

```
<xsd:complexType name="Abc">
   <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="Abc-item" type="xsd:integer"/>
   </xsd:sequence>
</xsd:complexType>
```

Any modifications to the basic element, such as restrictions on the size of a numeric variable or the length of a string, are handled as described in prior sections. Likewise, any bounds to the number of members of the sequence are handled as previously described.

If the sequence is such that tags for each item in the sequence are not necessary or desired, such as a sequence of lane numbers, and the elements in the sequence are simple elements with no internal spaces,[5] then the sequence may be defined as a space-separated list.

For example, replace `Abc ::= SEQUENCE OF INTEGER`: by:

```
<xsd:simpleType name="Abc">
   <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

5. Spaces and any other illegal characters shall be replaced with a a "–" character.

4.13 Defining Elements as Instances of Existing Basic Types—To define an element corresponding to an identifier for an existing ASN.1 basic type that corresponds to an XML basic type, replace xyz basictype with:

```
<xsd:element name="xyz" type="xsd:basictype">
```

The list of XML basic types is presented in Table 1. ASN.1 basic types found in Table 2 should be treated as defined in subsequent sections. If the basic type is an integer type, Table 3 of integer subtypes may be used.

For example, the ASN.1 definitions:

```
price-TimeInterval INTEGER (0..65535)
weather-Temperature INTEGER (-127..127)
directoryEntry-ExtendedInfoAvail BOOLEAN
```

Would be converted to:

```
<xsd:element name="price-TimeInterval" type="xsd:unsignedShort"/>
<xsd:element name="weather-Temperature" type="xsd:byte"/>
<xsd:element name="directoryEntry-ExtendedInfoAvail" type="xsd:boolean"/>
```

Examples of element content using these would be:

```
<price-TimeInterval>120</price-timeInterval >
<weather-Temperature>65</weather-Temperature >
<directoryEntry-ExtendedInfoAvail> True</directoryEntry-ExtendedInfoAvail>
```

For the most part, such elements are used in messages and data frames where very simple data concepts are needed and where the likelihood of reuse is small. If the data concept in question is likely to be reused in other places, a formal ASN.1 type definition should be used in the standard which defines it.

4.14 Defining Elements as Instances of User-Defined Types—To define an element corresponding to an identifier for a type that has been separately defined in the schema, replace xyz Definedtype with:

```
<xsd:element name="xyz" type="Definedtype">
```

For example, the ASN.1 definitions:

```
origin              LocationReference
route-Identity      Route-Identity,
route-Name          Route-Name,
route-Description   Route-Description,
```

Are converted to:

```
<xsd:element name="origin" type="LocationReference"/>
<xsd:element name="route-Identity" type="Route-Identity"/>
<xsd:element name="route-Name" type="Route-Name"/>
<xsd:element name="route-Description" type="Route-Description"/>
```

Examples of content using these would be (the chosen elements are all simple ones based on text strings):

```
<route-Identity>123a</route-Identity>
<route-Name>Downtown to Valley Mall Loop</route-Name>
<route-Description>Stops at all major cross street intersections,
      Buses run every 15 minutes from 7 AM to Midnight, all days
      </route-Description>
```

4.15 **Defining Elements as Instances of In-Line Defined Types**—Often a type is implicitly defined in an ASN.1 definition. That type is not available to be used in the definition of any other element or type. The same is true in XML definitions. In this case, the element is defined to be of the unnamed simple or complex type whose definition follows immediately. For these cases, the definitions of the unnamed simple or complex type follows the same rules as those defined for named types, presented in 4.1 through 4.12.

For example, given the ASN.1 definitions (taken from within another structure):

```
trip-MilesToNextManeuver  INTEGER        (1..999999)
traveler-Pager            NumericString (SIZE(1..20))
```

the corresponding XML definitions are:

```
<xsd:element name="trip-MilesToNextManeuver">
   <xsd:simpleType>
      <xsd:restriction base="xsd:unsignedInt">
         <xsd:minInclusive value="1"/>
         <xsd:maxInclusive value="999999"/>
   </xsd:simpleType>
</xsd:element>
<xsd:element name="traveler-Pager">
   <xsd:simpleType>
      <xsd:restriction base="NumericString">
         <xsd:maxLength value="20"/>
         <xsd:minLength value’"1"/>
   </xsd:simpleType>
</xsd:element>
```

Example of element content using these would be:

```
< trip-MilesToNextManeuver>150</ trip-MilesToNextManeuver>
<traveler-Pager>7036521157</traveler-Pager>
```

4.16 Defining BIT STRING Elements—For an element of BIT STRING type, one of two approaches shall be used. BIT STRINGs longer than 32 bits or of unspecified length are to be represented as a string of character 1's and 0's. For a type defined with 32 or fewer bits, define a new type as an enumeration, listing the text value corresponding to each bit. For any undefined bits, a text value is defined as "bn", where n is the position of the undefined bit. Then define the original type as a space-separated list containing elements of the type just defined. As an option, a character string of 0's and 1's may also be permitted, using a union construction.

These two options are illustrated below.

If the BIT STRING is longer than 32 bits or if its length is not defined, replace xyz BIT STRING by:

```
<xsd:element name="xyz" type="binary"/>
```

where the type binary is defined elsewhere as:

```
<xsd:simpleType name="binary">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[01]*"/>
   </xsd:restriction>
</xsd:simpleType>
```

If the values of all the bits are defined, replace xyz BIT STRING by:

```
<xsd:simpleType name="xyz-item">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="bit 1 text"/>
         <xsd:enumeration value="bit 2 text"/>
      . . .
         <xsd:enumeration value="bit n text"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="xyz">
      <xsd:list itemType="xyz-item"/>
</xsd:simpleType>
```

If it is desirable to allow a character BIT STRING representation, use the union construction:

```
<xsd:simpleType name="xyz-item">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="bit 1 text"/>
         <xsd:enumeration value="bit 2 text"/>
         . . .
         <xsd:enumeration value="bit n text"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="xyz">
      <xsd:union>
         <xsd:simpleType>
            <xsd:list itemType="xyz-item"/>
         </xsd:simpleType>
         <xsd:simpleType>
            <xsd:restriction base="binary"/>
         </xsd:simpleType>
      </xsd:union>
</xsd:simpleType>
```

If the values of some bits are not defined, e.g., bits 3,4, and 7, then enumeration text following the below pattern shall be inserted in the sequence:

```
<xsd:enumeration value="b3"/>
<xsd:enumeration value="b4"/>
<xsd:enumeration value="b7"/>
```

For example, given the ASN.1 of:

```
Day-of-week ::= BIT STRING (SIZE(8))
     holiday   Day-of-week ::= '10000000'B
     sunday    Day-of-week ::= '01000000'B
     monday    Day-of-week ::= '00100000'B
     tuesday   Day-of-week ::= '00010000'B
     wednesday Day-of-week ::= '00001000'B
     thursday  Day-of-week ::= '00000100'B
     friday    Day-of-week ::= '00000010'B
     saturday  Day-of-week ::= '00000001'B
```

The resulting schema would be:

```
<xsd:simpleType name="Day-of-week-item">
   <xsd:restriction base="xsd:string"/>
      <xsd:enumeration value="holiday"/>
      <xsd:enumeration value="Sunday"/>
      <xsd:enumeration value="Monday"/>
      <xsd:enumeration value="Tuesday"/>
      <xsd:enumeration value="Wednesday"/>
      <xsd:enumeration value="Thursday"/>
      <xsd:enumeration value="Friday"/>
      <xsd:enumeration value="Saturday"/>
   <’xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="Day-of-week">
   <xsd:list item-type="Day-of-week-item"/>
</xsd:simpleType>
```

Suppose scheduleDay is defined as:

```
<xsd:element name="scheduleDay" type="Day-of-week"/>
```

Then an example of content using this would be:

```
<scheduleDay>Monday Tuesday Wednesday</scheduleDay>
```

4.17 Defining OCTET STRING Elements—For an element defined as an OCTET STRING, replace xyz OCTET STRING by:

```
<xsd:element name="xyz" type="OctetString"/>
```

Where octetString has been defined elsewhere in the schema with:

```
<xsd:simpleType name="OctetStringOptions">
```

```
      <xsd:union memberTypes="xsd:hexBinary xsd:base64Binary"/>
   </xsd:simpleType>
   <xsd:complexType name="OctetString">
      <xsd:simpleContent>
         <xsd:extension base="OctetStringOptions">
            <xsd:attribute name="EncodingType" use="required">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:NMTOKEN">
                     <xsd:enumeration value="hexBinary"/>
                     <xsd:enumeration value="base64Binary"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
```

For purposes of converting ASN.1 standards to XML schemas, it is recommended that the use of octet strings be kept to a minimum, and that wherever possible, an element should be defined as a character string rather than an octet string.

Any length restrictions to an element defined as an OCTET STRING may be handled with maxLength, minLength, and length restrictions as described in 4.4 and 4.5 – with the resulting conversion length being used. Observe that the conversion of an octet string into one of these formats typically changes its length, and any size constraint found in the ASN.1 must be expressed in the resulting schema reflecting the new length.

For example, given the ASN.1 of:

```
DatabaseNumber                 ::= OCTET STRING (SIZE(0..12))
DirectoryEntry-ExtendedInfo    ::= OCTET STRING (SIZE(1..4194303))
```

The resulting schema would be:

```
<xsd:element name="DatabaseNumber">
   <xsd:simpleType base="OctetString">
      <xsd:restriction>
         <xsd:minLength value="0"/>
         <xsd:maxLength value="12"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
<xsd:element name=" DirectoryEntry-ExtendedInfo ">
   <xsd:simpleType base="OctetString">
      <xsd:restriction>
         <xsd:minLength value="1"/>
         <xsd:maxLength value="4194303"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

Example of content using hexBinary encoding follow:

```
<DatabaseNumber EncodingType = "hexBinary">1234</DatabaseNumber>
<DirectoryEntry-ExtendedInfo EncodingType ="hexBinary">52nf37n6483ned3de
         </DirectoryEntry-ExtendedInfo>
```

In some instances in the standards, the use of an OCTET STRING has been employed as a placeholder for a definition to be found in another standard. Regrettably, the documentation of this presumed linkage is often very poor. Implementers of these conversion rules should seek to understand the intent of the original ASN.1 and if that intent was to refer to another standard's data elements, that definition should be employed rather than converting as an OCTET STRING type. Another use of an OCTET STRING has been to denote the mixed use of text (IA5String types) and codes, with the content encoding following the rules of SAE J2540. When this type of definition is found the conversion recommendations for sequences of mixed text and codes shall be followed.

Refer to 5.2 for details.

4.18 NULL Elements—For an element defined as an NULL type, replace `xyz NULL {. . . }` by:

```
<xsd:element name="xyz">
</xsd:element>
```

Although NULL types are permitted in ASN.1 there are no examples of its use in SAE standards definitions.

4.19 OBJECT IDENTIFER (OID) Elements—Object identifiers have no purpose in XML. Elements are identified using XLINK or XPATH by their tag names and their positions within the document hierarchy, not by object identifiers. If it is nevertheless desired to keep the OID definitions in the XML schema, they may be treated as optional elements with string value. For an element defined as an OBJECT IDENTIFER OID type, replace xyz OID {. . . } by:

```
<xsd:element name="xyz" type="xsd:string"/>
```

For example, given the ASN.1 (taken from a message in the IEEE Incident Management Message set, 1512) of:

```
msgOID OBJECT IDENTIFIER OPTIONAL,
```

The resulting schema would be:

```
<xsd:element name="msgOID" type="xsd:string"/>
```

And an example of content using this would be:

```
<msgOID>536.26235.12.6.1</msgOID>
```

4.20 OPTIONAL Elements—For any element that is OPTIONAL when used in a structure, the OPTIONAL keyword is represented in XML as a constraint facet by adding `minOccurs="0"` inside the element definition tag:

For example, given the ASN.1 fragment of the ResponseType message:

```
pollution-SmogAlert        INTEGER (0..255) OPTIONAL,
pollution-AirQualityIndex INTEGER (0..255) OPTIONAL,
pollution-CarbonMonoxide  INTEGER (0..255) OPTIONAL,
```

The resulting schema fragment would be:

```
<xsd:element name="pollution-SmogAlert"
             type="xsd:unsignedByte" minOccurs="0"/>
<xsd:element name="pollution-AirQualityIndex"
             type="xsd:unsignedByte" minOccurs="0"/>
<xsd:element name="pollution-CarbonMonoxidet"
             type="xsd:unsignedByte" minOccurs="0"/>
```

4.21 Handling Comments—When a comment is to be used the following rules shall apply: For comments which are short (one or two lines long at most) the XML comment tag shall be used. Comments which are longer shall be enclosed in the <documentation> tag within an <annotation> tag with the leading "—" removed. The leading "—" shall be preserved if the comment appears in an appinfo tag after mapping contents, thereby preserving the comment nature of the line and to assist those that may wish to process appinfo tag content in a fashion similar to lines of valid ASN.1.[6] Content of the text shall employ the normal XML exception encoding (e.g., "<" becomes "&lt" etc.) at all times. Placement shall follow the original ASN.1 to the degree practical. When a compact schema is created for operational purposes, it should be possible to strip the comments from it.

For example, given the ASN.1 fragments in the following examples:

```
-- LSB units of 1/10th of a mile
```

and:

```
-- HH=00 through 23; MM=00 through 59;
-- SS=00 through 59; ssss=0000 through 9999.
-- HH represents hours, MM minutes, SS seconds,
-- and ssss decimal seconds to whatever number
-- of significant digits is required
```

6. Refer to 4.6 and 5.1 for further recommendations on the use of this tag type to preserve ASN.1 relationships.

The resulting schema portions for these fragments would be:

```
<!-- LSB units of 1/10th of a mile  -->
```

and:

```
<xsd:annotation>
   <xsd:documentation>
      HH=00 through 23; MM=00 through 59;
      SS=00 through 59; ssss=0000 through 9999.
      HH represents hours, MM minutes, SS seconds,
      and ssss decimal seconds to whatever number
      of significant digits is required
   </xsd:documentation>
</xsd:annotation>
```

4.22 Commonly Repeated Element—Any element which is defined the same way in several places may be defined once in the schema and referred to with a ref=”…” clause anywhere else in the document. However, the SAE ATIS committee decided not to do this because the overriding ITS operating principle of preferring a formal ASN.1 Type Definition provided a better alternative. It is recommended that any such common definitions should be declared as a formal type definition in the defining ASN.1, and subsequent element definitions may use the defined type in all messages.

When this is done, any valid ASN.1 name may be used for the new type.[7] Two additional benefits of this approach are that (1) it is consistent with the approach of the ITS Data Registry, and (2) it enables the defined type to be used by other schemas when imported using a namespace declaration. Recent revisions of SAE standards have moved in this direction. The common types included in the SAE J2354 message set are listed in Section 6.

4.23 Defining an Element of a Type Defined in a Different Schema—If an element is defined as belonging to a type is defined in a different namespace (that is, if the ASN.1 definition comes from another standard), then:

a. Define a prefix to refer to that namespace as per 5.9
b. Import the namespace into the base schema
c. Use the defined prefix with a colon before the name of the imported type

For example, if SpecialType is defined in the WXYZ namespace defined in another standard, and the element specialType is defined to be of that type then:

a. Include the following attribute and value in the initial schema element:

```
xmlns:wxyz "URL-for-WXYZ-namespace"
```

b. Import the namespace, giving its name and its location:

```
xsd:import namespace="URL-for-WXYZ-namespace"
           schemaLocation="address-of-WXYZ-schema"
```

7. In the initial adopted message set standards of SAE and ITE there were/are a number of times when the data elements (defined in other adopted standards) are repeated in-line rather than referred to as formal types. In some of these there exist errors or numbering variations between various lists. [This is prevented and overcome by using the type definition method now preferred for new and revised work.] When such a discrepancy exists in a referenced standard, the adopted data dictionary entry defining the element has been used. It should be stressed that it is the responsibility of the defining SDO (not the SAE) to resolve such discrepancies and issue definitive advice to others.

c. Define element specialType as:

```
<xsd:element name="specialType" type="wxyz:SpecialType/>
```

4.24 Defining Top-Level Elements—Top-level elements need to be created for each message type permitted by the schema. A strict translation of the ASN.1 definition to a schema provides a type for each message type, but not an element instance of the type than can be used as the root element of a conformant message. Therefore, for each complex type that is a message type, create a top-level element of that type.

For example for message type:

```
<xsd:complexType name="TravelerInformationRequest">,
```

create a top-level element

```
<xsd:element name="TravelerInformationRequest"
             type="TravelerInformationRequest">.
```

These global element definitions should go at the beginning of the schema.

4.25 Beginning and Ending the Schema—Start the schema with:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Close the schema with:

```
</xsd:schema>
```

Append a final comment line with the date the schema was constructed, a suitable reference to the standard, and the precise revision from which it came in the form of a comment <!-- Text -->

5. Discussion of Selected Conversion Rules—Most of the rules for converting ASN.1 definitions to XML definitions are quite straightforward. In general the translation process is one to one, that is, each ASN.1-type definition is replaced by one XML schema element definition. This section discusses several of the conversion rules presented in the previous section that are not obvious. Topics include the treatment of enumerated elements, sequences of these elements often found in the ITIS lists used with text, the treatment of bit string types, and the treatment of binary content (octet strings).

The rationale for needing these rules is primarily due to the differences in technical approach between ASN.1 and XML. The only exception to this is the handling of sequences of ITIS lists mixed with text where a more complex structure is required. Finally rules concerning the use of XML namespaces to sort out references to element types defined in other standards are presented.

5.1 Methods of Defining ENUMERATED Elements—The method of defining enumerated types originally proposed by members of the SAE ATIS committee in April 2001 (allowing either an integer index or an empty element whose tag name is the text value of the element) has the following problems:

a. It cannot be implemented in XML without declaring the element to be of mixed type, thus losing any control over the value of the element content
b. The value represented by the enumerated textstrings should be the content of the element, not the name of an additional empty element
c. It does not take advantage of the XML enumeration capabilities

The following method of defining enumerated elements in an XML schema uses the union construction to permit either an integer in the specified range or a text string with one of the enumerated values to appear as the content of the enumerated element.

As an example from the SAE J2354 standard, the following ASN.1 declaration:

```
Trip-GuidanceLevel   :== ENUMERATED  {
   maneuversOnly           (0),
   linksBetweenManeuvers   (1),
   connectedLinks          (2),


...
}
-- values up to 127 reserved for std use
-- values from 128 to 255 for local use
```

would produce the following XML schema declaration:

```
<xsd:element name=" Trip-GuidanceLevel ">
   <xsd:simpleType>
      <xsd:union>
         <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
               <xsd:minInclusive value="0"/>
               <xsd:maxInclusive value="2"/>
            </xsd:restriction>
         </xsd:simpleType>
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="maneuversOnly"/>
               <xsd:enumeration value="linksBetweenManeuvers"/>
               <xsd:enumeration value="connectedLinks"/>
            </xsd:restriction>
         </xsd:simpleType >
      </xsd:union>
   </xsd:simpleType>
</xsd:element>
```

If there are no restrictions on the minimum or maximum values of the index beyond those imposed by the base type itself, the example above may be shortened somewhat by using the “memberType” attribute of the union tag, as follows:

```
<xsd:element name=" trip-GuidanceLevel ">
   <xsd:simpleType>
      <xsd:union memberTypes="xsd:integer">
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="maneuversOnly"/>
               <xsd:enumeration value="linksBetweenManeuvers"/>
               <xsd:enumeration value="connectedLinks"/>
            </xsd:restriction>
         </xsd:simpleType >
      </xsd:union>
   </xsd:simpleType>
</xsd:element>
```

The ASN.1 declaration above clearly associates the integer value 0 with the text value “maneuversOnly”, the integer value 1 with the text value “linksBetweenManeuvers” etc. There is no way to specify the same correspondence in an XML schema. In fact, that is not the job of the schema. The schema validation process confirms that the contents of the trip-GuidanceLevel element is either an integer (or an integer between 0 and 2 inclusive) or one of the enumerated text values. If it is necessary for an application program receiving an XML document including integer codes to supply the text corresponding to the received integer code, that is the job of the application program. To enable an application program to construct a correspondence table, and ensure that the correspondence table is part of the standard itself, the code-value definitions may be placed in the XML schema inside <annotation> and <appinfo> tags. Anything inside <annotation> and <appinfo> tags is ignored by the schema validation process, but may be used by any other application program that can read the schema. For the example above, the following lines are inserted into the schema:

```
<xsd:annotation>
   <xsd:appinfo>
      maneuversOnly (0)
      linksBetweenManeuvers (1)
      connectedLinks (2)
   </xsd:appinfo>
</xsd:annotation>
```

If the items in the list follow the ITS convention that the value zero is used to indicate that no suitable item is found in the list (and that a free text entry is expected to be found in the data frame which uses it and should be consulted) then the first line of the <xsd:appinfo> section should be as below unless a term is already defined for this value.

The ASN.1 declaration above makes use of the “…” clause to denote that the list may be extended at some future time.[8] This notation is an aspect of ASN.1 which was added so that various encoding rules used by some ASN.1 systems would not be caught off-guard if new items were added to a future revision of a structure. In ITS enumerations this clause is now routinely added to indicate that new items are expected in the future. Further, as an adopted convention within ITS many lists allow for the addition of “locally defined items” in the range of 128 to 255 for such lists. The addition of such items requires a re-definition of that portion of the schema and the new portion of the schema is typically declared in a namespace local to the implementation.

8. The important aspect is the word *future*. When such items are not currently in a list, there is no reason to extend the “maxInclusive” term above the items present.

The revision of a standard textual phrase for use on a local basis (e.g., “danger, ice on bridge” revised to become “warning, ice on bridge”) would also require such a local schema. See 4.6 for further details of how the construct is used.

5.2 **Alternate Methods of Defining Mixtures of Text Strings and ENUMERATED Elements—**An increasingly common occurrence in the message sets of the ITS industry is the mixture of open textual phrases (free text) with phrase codes, typically the ITIS phrases found in SAE J2540-2 representing common traffic events and conditions. Occurrences of this can be found in SAE and IEEE standards and are expected to be added to ITE work as well. These are defined in various ways in adopted messages, including as IA5Strings, as OCTET strings, and more recently as a SEQUENCE OF a defined type called “ITIS-Code” - in any event the intent in such cases is a collection of codes interspersed with textual content. The complete syntax encoding allowed by SAE J2540 is moderately complex and does not translate to XML in all its features as it was designed for both large and small bandwidth uses. The most common formats found involve simply the interspersing of entries from a single table (the ITIS table) with free text. This style of use fits extremely well with XML using the "mixed" content construct. A recommendation on how it is translated (from a string, octet or other format as the case may be) is as follows:

Given any of the following definitions (various types of this sort will be found in the current standards):

```
someMixedText  ::=  OCTETSTRING(SIZE(1..25))
someMixedText  ::=  IA5String (SIZE(1..25))
someMixedText  ::=  SEQUENCE OF ITIS-Codes (SIZE(1..25))
```

The XML would be:

```
<xs:element name="someMixedText">
   <xs:complexType mixed="true">
      <xs:sequence>
         <xs:element name="ITIS" minOccurs="0">
            <xs:simpleType>
               <xs:restriction base="xs:ITIS:code">
                  <xs:maxLength value="25"/>
                  <xs:minLength value="1"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
    </xs:element>
```

As an example of XML using these rules consider the phrase. *“Multi vehicle accident, no through traffic, exit at Maple Ave and use left hand parallel roadway.”* This can be expressed as three phrases and some free text as: Multi vehicle accident [ITIS code 517] no through traffic [ITIS code 2571], exit at Maple Ave and [some free text] use left hand parallel roadway [ITIS code 7193]. This is rendered in XML as:

```
<someMixedText>
   <ITIS> Multi vehicle accident  </ITIS>,
   <ITIS> no through traffic </ITIS>
          , exit at Maple Ave and
   <ITIS> use left hand parallel roadway </ITIS>.
</someMixedText>
```

Less desirable, but also valid, would be rendered using the code values rather than the enumerated text values as:

```
<someMixedText>
   <ITIS> 517 </ITIS>,
   <ITIS> 2571 </ITIS>
          , exit at Maple Ave and
   <ITIS> 7193 </ITIS>.
</someMixedText>
```

Observe that the comma and period are also free text in the above example. If white space should be preserved, replaced or collapsed in the final rendering, then the proper option [e.g., <whiteSpace value='preserve'/>] may also be set in the restrictions.

If the mixture of text and code involves a different set of codes than ITIS, a unique tag shall be used for each unique set of code content. If the codes used are the SAE RDS codes defined in SAE J2540-1 then the tag <RDS> shall be used. If the codes used are the SAE Street Name codes defined in SAE J2540-3 then the tag <StName> shall be used. If only a subset of a code set (such as from ITIS) is being used, then the same tag as the code set shall be used. Observe that a namespace may be useful for long lists of codes. If a name space is used it shall follow the naming rules defined elsewhere in this document.

Observe that if the need to render ITIS elements to a stream of untagged text is required, a simple style sheet can be constructed to remove the intervening tags. At the same time, those wishing to automatically sort and process a message containing ITIS elements searching for keywords / codes can also proceed with a style sheet approach. If the textual enumerations for the ITIS content are used, the result is also fairly humanreadable, a key benefit to some XML users. Either of these needs can also be handled by in-line processing as the message arrives if the construction of a full object model of the message is not suitable for the application. Aside, this was one of the reason the use of the entities[9] construct was not employed.

Finally, observe that not all octets or strings are intended to hold mixed phrase code and text content. The translator must be aware of the intent of the committee in order to make the correct determination.

5.3 Alternate Method of Defining BIT STRINGSs—The most typical reason for defining an element as a BIT STRING rather than as an enumerated element is to allow more than one value to be specified at the same time[10]. For example, days in a bus schedule or weather conditions or road conditions may have multiple values set to true. The following rule for permitting multiple elements or a character bit string shall be used. The example is the definition of ATIS-DayOfWeek.

The first step is to declare a simple XML type enumerating the text values. The second step is to declare the original element as a union of two simple types. The first possible type is a list of the enumerated element types. The second possible type is a binary character string.

9. Entities are an XML construct for named fragments of content that can be used somewhat like an include file or a processing macro to expand into the content before the document is processed further. However, a complete document is typically required, and this is at times cost prohibitive to small applications that wish to monitor the stream of data without processing it further until selected keywords of content are seen.
10. Common practice in recent ITS efforts has been to reduce the use of the *BIT STRING* type in favor of using a *SEQUENCE OF* and an *ENUMERATED* type the achieve the same results.

```
<xsd:simpleType name="ATIS-DayOfWeek-item">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Sunday"/>
      <xsd:enumeration value="Monday"/>
      <xsd:enumeration value="Tuesday"/>
      <xsd:enumeration value="Wednesday"/>
      <xsd:enumeration value="Thursday"/>
      <xsd:enumeration value="Friday"/>
      <xsd:enumeration value="Saturday"/>
      <xsd:enumeration value="includeHolidays"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ATIS-DayOfWeek">
   <xsd:union>
      <xsd:simpleType>
         <xsd:list itemType="ATIS-DayOfWeek-item"/>
      </xsd:simpleType>
      <xsd:simpleType>
         <xsd:restriction base="binary"/>
         <xsd:length value="8"/>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>
```

The methods described above for representing BIT STRINGS and enumeration may be combined. For example, if `DayOfWeek-listitem` above were defined as:

```
   <xsd:simpleType name="ATIS-DayOfWeek-item">
      <xsd:union>
         <xsd:simpletype>
            <xsd:restriction base="xsd:string">


            <xsd:enumeration value="Sunday"/>
            <xsd:enumeration value="Monday"/>
            <xsd:enumeration value="Tuesday"/>
            <xsd:enumeration value="Wednesday"/>
            <xsd:enumeration value="Thursday"/>
            <xsd:enumeration value="Friday"/>
            <xsd:enumeration value="Saturday"/>
            <xsd:enumeration value="includeHolidays"/>
         </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
         <xsd:restriction base="xsd:unsignedByte">
            <xsd:maxInclusive value="7"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:union>
</xsd:simpleType>
```

Then `ATIS-DayOfWeek` could appear as a space-separated list of text words or a list of numbers, or a mixture of both.

5.4 Treatment of IA5String Contents—Commonly found in ASN.1 encoding are IA5String type strings, consisting of characters in the range from zero to 255 hex, one per octet. However, since XML is primarily a character-based approach, any bytes containing non-character sequences must use special escape characters. Some of these characters have special meaning in XML (for example "&" or "<") and must be replaced with the escape characters found in the XML standard (e.g., "&" is replaced by "&amp"). When a subset of this character range is used (for example the numeric strings) then the same rules apply. In a number of existing ITS standards the IA5String is employed in the actual ASN.1 definitions, but with comments implying that the ASCII character set is to be used. In such cases it shall be up to the issuing committee to resolve any ambiguity or conflict.

5.5 Treatment of OCTET STRINGs—Since ASN.1 encoding is a binary encoding, OCTET STRINGs consisting of arbitrary strings of bits do not present any problem. However, since XML is primarily a character-based approach, any bytes containing noncharacter sequences must use special escape characters, or be represented as hex characters. For purposes of converting ASN.1 standards to XML schemas, it is recommended that the use of OCTET STRINGs be kept to a minimum, and that wherever possible, an element should be defined as a character string rather than an octet string. When an OCTET STRING must be used, conversions to several suitable industry formats are presented in Section 4. Observe that the conversion of an octet string into one of these formats typically changes its length, and any size constraint found in the ASN.1 must be expressed in the schema reflecting the new length.

It should also be noted that in recent committee message definitions work, large binary files of this type are often being "pointed to" with an in-line URL/URI rather than being included in-line themselves. This is the preferred embodiment if XML is to be used as the conversion process, because placing the binary content inline is wasteful of bandwidth when compared to the use of the pointer method.

5.6 Treatment of Manual ASN.1 Tags—While the syntax expression of ASN.1 is an internationally accepted standard and used in most ITS standards work, the encoding of ASN.1 for actual transmission allows many encoding flavors to fit different needs (BER, OER, PER, etc.) and each has a number of proponents and opponents to its use. As a function of an international treaty, the US has used ASN.1 as the syntax of choice in its work. Those that have chosen to use ASN.1 encoding for transmission have at times had to become involved in the manual selection and maintenance of tags to be used. This type of ASN is recognizable by the "x]" elements in the source code where x is the tag value itself.

There is nothing in XML analogous to these tags and no useful method for preserving them is therefore needed. However, there have been some prior efforts to keep the use of the tags in some draft TMDD messages in alignment with parts of the ITIS codes which are a part of the SAE J2540.2 standard. In this effort the upper byte of these two-byte codes was chosen to be the exact same value as the tag to be used when that specific sub-list of elements was expressed in ASN.1 OER encoding. The lower byte became the value, selecting the item in the list. By this method the implementer could view the "bits on the wire" representing the list as a long list of many useful phrases (2 bytes long selecting the entry) or as a series of short lists with the same entries (first byte selects the list, second byte selects the entry). The organization of the list and meaning of the codes remained the same (which was the key issue for interoperability to occur).

In XML there are no such numeric tags; rather, a textual tag is used derived from the element name. As a result the full 16 bit value (or its textual value) is always used for an item. The range constraint, often found when the ITIS phrase list is used (used to limit to a suitable subset), then becomes the XML min and max values. If multiple ranges are used, a union is employed to reflect them all. Under no circumstances shall the lower byte of a two-byte code list be used in isolation; the upper byte would also always be present.

5.7 Treatment of Long ASN.1 Names—Because the naming convention used in ASN.1 are not then used in any format sent over the transmission media, there is no penalty for employing long names in type definitions. As a result of this, in some message set families, extremely long names can be found which employ embedding naming conventions in the ASN.1 names (more properly called descriptive names in the current data registry). Current practice remains divided among the ITS messages effort, but in general shorter names are now preferred for the ASN.1 with the descriptive naming convention limited to application in that part of the registry entry. Conversely, in XML, the name of the element or the instance of the element is most often used to form the name of the tag to be used.

This can result in overly long names which are problematic. The recommendation is that conversions should use that name of the ASN.1 element when possible but may select a shorter and concise name for the tag when this occurs. Those making this determination should bear in mind that the length of a long tag must be transmitted every time that tag is sent, so from an efficiency perspective short tag names should be chosen.

5.8 Treatment of ASN.1 Comments—As a general principal a working schema should be kept short and comments should be constrained, but a schema that serves as an official definition of a message set should have a full set of comments. A verbose document requires greater bandwidth to transfer and if transmission of the schema set must occur routinely for content validation, then the cumulative effect is reduced performance.[11],[12] In the ASN.1 document, verbose comments can often be found, and as verbosity in this area does not affect performance (but promotes understanding), it is to be encouraged.

It is acceptable to create a schema (or an XML document) with verbose comments for the purpose of standards definition or teaching aid, but such a document should never be used in a production system. Such a schema should be marked with namespace words such as “training” or “commented” to distinguish it.[13] Such a verbose schema shall contain a comment pointing the user to the terse version if one exists. Such a verbose schema, once stripped of comments and documentation tag content, should result in the same schema as the terse one adopted by the standards body.

Short comments shall be placed in the document where needed using the <!-- --> format. When longer comments are to be used in the schema the <documentation> tag within the <annotation> tag structure shall be used.

For example; the common ASN.1 comments found at the end of many enumerated lists:

```
-- values to 127 reserved for std use
-- values 128 to 255 reserved for local use
```

11. As with the issue of long names, the premise of XML is to be self-describing of the structure between users on the fly and with no prior coordination. This is in contrast to ASN where all parties are presumed to have acquired the actual ASN and produce their own derivative works from it before communications begin. Therefore as a goal we strive to keep the files exchanged as small as possible.
12. Many industries that have embraced XML as a panacea are now experiencing turnmoil because they lacked any internal agreement regarding the content they wished to exchange by way of XML. The ASN.1 standards, and multi-year committee process that developed them, has allowed the ITS industry to avoid much of this issue.
13. For example in the training and outreach effort SAE has produced some trivial software tools that produce XML and which allow the user to select a verbose mode that then adds usage commentary interspersed with each expression. Very useful for teaching, but not something one should do on a wide scale.

Would be expressed as:

```
<xsd:annotation>
   <xsd:documentation>
      values to 127 reserved for std use
      values 128 to 255 reserved for local use
   </xsd:documentation>
</xsd:annotation>
```

A comment containing information on the relevant revision of the standard used and the date which the schema was created shall be added.

5.9 Use of XML Namespaces to Point to Types Defined in Other Standards—Many of the data element types used in the ATIS standards are drawn from other standards sets, including the Traffic Management Data Dictionary (TMDD), the Transit Communications Interface Profiles (TCIP) standard, and the Location Reference Message Set (LRMS) and the recently completed SAE ITIS phrase effort. This trend of shared data elements across all the message sets is expected to grow over time as the data registry harmonization process continues.

In order to use XML with these standards it is necessary to develop a schema for those parts of the external standard which are to be used. To preserve the connection to these other standards and pave the way for future coordination of the various transportation standards, XML namespace capabilities were used to define an integrated package. Here we propose a simple namespace format to be used by all the existing ITS message based on the four letter acronyms for them. The following rule expresses how this shall be done:

a. Separate schemas were constructed containing data types referenced by the ATIS standard from other standards as well as those developed and maintained by the SAE. Thus the following schemas were developed by the SAE effort which used these rules:

   LRMS.XSDAll content dealing with LRMS structures used by SAE-ATIS and other SDOs
   ITIS.XSDAll SAE J2350.2 “ITIS” Phrase lists used by SAE-ATIS and other SDOs
   ATIS.XSD All other message and data content defined in SAE ATIS messages, SAE J2353/54/69/et al
   TMDD.XSD Those portions of the TMDD standards used by SAE-ATIS
   TCIP.XSD Those portions of the TCIP standards used by SAE-ATIS

   The first three schema will be formally adopted by the SAE standards process and their governing committees. These schema contain the entire adopted message set and represent the committee recommendation on how these message should be expressed in XML.
   The last two schemas contain only the types referenced by the ATIS standard and follow the same rules as the prior schemas. These will be placed as an informative[14] annex to the ATIS.XSD document so that implementers have a complete set of elements from which to build.
   The last two schemas may eventually grow to contain the entirety of those standards, but for now they contain only the types referenced by the ATIS standard. It is anticipated that this effort will be undertaken by each of the standards organizations responsible for the subject standard following a time and pace of that organizations choosing. Upon completion of this effort by each group it is expected that the schema produced will reflect both the ASN.1 of that standard and the translation rules adopted by that group and will supercede the informative annexes mentioned above.

14. In other words, these appendicies will NOT be a standard of the SAE.

b. In addition, the following namespaces are proposed for other areas of ITS message which it is known will be translating into XML schemas in the near future. Thus the following namespaces are proposed:

   IM.XSD All content dealing with the IEEE Incident Management 1512 family of standards
   DSRC.XSD All content dealing with the ASTM & IEEE DSRC message efforts
   ITS.XSD Reserved for the construction of an ITS wide common data registry repository
   XXXX.XSD Who else should we add at this time?????

c. Any possible use of up to the next four letters in each of above namespaces is expressly reserved for use by the SDOs owning the subject message set. It is intended for the SDOs, version control and management needs and shall not be used by others. For example, ATIS_B or TMDDrev5 would be prohibited. [Note that the target namespace can be anything desired]
   In use, each of these subsidiary schemas is defined with a default namespace and a target namespace. The default namespace is the namespace used for all defined types and elements that are not explicitly qualified in the schema with a namespace prefix. The target namespace is the namespace created when the schema is compiled or imported. For ATIS schema purposes, the two namespaces are the same.
   Following current Internet conventions, the namespaces are defined in the form of a Uniform Resource Locator (URL), but they do not represent actual locations on the web. The third namespace defined at the start of each schema is the W3C namespace which defines elements and types used in constructing the schema itself.
   For example, the opening schema element for the TMDD schema is:

```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.dummy-tmdd-address"
xmlns="http://www.dummy-tmdd-address">
```

   The ATIS schema ATIS.XSD first defines the prefixes it will use to refer to the other namespaces…

```
xmlns:lrms="http://www.dummy-lrms-address"
xmlns:itis="http://www.dummy-itis-address"
xmlns:tcip="http://www.dummy-tcip-address"
xmlns:tmdd="http://www.dummy-tmdd-address"
```

   …and then imports the namespaces, specifying the name of the namespace and the name of file (in the same directory as ATIS.XSD) where the schema is located.

```
<xsd:import namespace="http://www.dummy-lrms-addr" schemaLocation="lrms.xsd"/>
<xsd:import namespace="http://www.dummy-itis-addr" schemaLocation="itis.xsd"/>
<xsd:import namespace="http://www.dummy-tcip-addr" schemaLocation="tcip.xsd"/>
<xsd:import namespace="http://www.dummy-tmdd-addr" schemaLocation="tmdd.xsd"/>
```

   When the ATIS schema is compiled or validated, the additional schema files are imported, and the types they contain are available using the prescribed namespace prefixes.

d. In the body of the ATIS schema, whenever a type is used from one of the imported namespaces, the type name is prefixed by the prefix specified in the xmlns command. For example:

```
<xsd:element name="trip-OriginLocation" type="lrms:LocationReference"/>
<xsd:element name="link-Status" type="tmdd:Link-status" minOccurs="0"/>
```

e. It can at times prove necessary in local implementations to add to the message sets, modify selected elements, add item to lists, or in other ways change the schema definitions produced by the standards bodies.

When this is required, one or more local namespaces following the same rules shall be used. The contents of these schemas should rely on the national schema as much as possible and should expressly avoid redefining the schema contents except as a means to extend or to patch definitions for local needs. It is acceptable and encouraged to use some form of date stamp or revision in the naming.

5.10 Expressing the Resulting Schema in a Standard—Within the SAE process any schema which is developed from these rules will be published in two ways once adopted by the normal SAE ASNI process rules. First, the document[15] which has the defining ASN.1 will also have the schema itself balloted at the same time. It is anticipated that just as individual data elements and messages are presented in the documents, “one per section”, that the XML schema for that portion will also be presented. This results in a more readable document with each section containing an ASN.1 entry and an XML entry alongside the usage discussion text for that item. At the end of the document the complete schema will be presented for completeness. An on-line web version of the schema will also be made available for validation use. Verbose variants may also exist as part of the training effort.

The process of expressing the schema resulting from applying these rules will vary by the conventions of each SDO and the needs of its members. The SDO should be consulted for definitive information on their own process. SDOs are strongly urged to make the schema available in an electric format to avoid data entry errors.

6. New Types Defined for the SAE ATIS J2353/2354 Schema—Most of the types defined in the XML version of the SAE J2353/2354 standard are direct conversions of the types defined in the ASN.1 version of the standard, using the rules presented in Section 4. A few new types were added, however, to implement special conversion rules, to avoid duplication of common in-line type definitions, or to substitute for a missing type definition. This section documents the additional type definitions.

Their types are also recommended to be used in other schemas.

6.1 Binary Type—Name: binary

Definition:

```
<xsd:simpleType name="binary">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[01]*"/>
   </xsd:restriction>
</xsd:simpleType>
```

Reason: After deliberation it was agreed[16] to allow bitstrings to be represented by a sequence of character 0’s and 1’s. This method of specifying bit strings as characters is retained as an option in the final rule set.

Example:

```
<xsd:element name="trip-PreferenceType" type="binary"/>
```

15. At the time of writing this would be parallel effort revising SAE J2554 to update and contain almost all SAE-ATIS content. However, new documents with other numbers and content are likely to exist in the future.
16. At the April 3rd 2001 meeting in Washington. However, textual values representing the measning of each bit position are preferred for readability, see 6.18.

NOTE— Every element that was originally defined as a bit string in the October 2000 J2353 standard is better represented as an enumerated type, or a list of elements of enumerated type, so that there are no good examples of purely binary or bit string elements currently in the SAE standards.

6.2 OctetString Type—Name: OctetString

Definition:

```
<xsd:simpleType name="OctetStringOptions">
   <xsd:union memberTypes="xsd:hexBinary xsd:base64Binary"/>
</xsd:simpleType>
<xsd:complexType name="OctetString">
   <xsd:simpleContent>
      <xsd:extension base="OctetStringOptions">
         <xsd:attribute name="EncodingType" use="required">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="hexBinary"/>
                  <xsd:enumeration value="base64Binary"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

Reason: This is a provisional definition of an octet string, able to represent non-printable characters either as binary or as hex characters. It is retained as an option in the final rule set.

Example:

```
<xsd:element name="device-Identity" type="OctetString"/>
```

6.3 NumericString Type—Name: NumericString

Definition:

```
<xsd:simpleType name="NumericString">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d*"/>
   </xsd:restriction>
</xsd:simpleType>
```

Reason: Telephone numbers are represented as numeric strings, but these are not a basic XML type.

Example:

```
<xsd:element name="traveler-Phone">
   <xsd:simpleType>
      <xsd:restriction base="NumericString">
         <xsd:length value="15"/>
      </xsd:restriction>
   <xsd:simpleType>
</xsd:element>
```

6.4 **BoundedString Type—**Name: BoundedString

Definition:

```
<xsd:simpleType name="BoundedString">
   <xsd:restriction base="xsd:string">
      <xsd:maxLength value="255"/>
      <xsd:minLength value="1"/>
   </xsd:restriction>
</xsd:simpleType>
```

Reason: Many variables throughout the schema were defined as: IA5STRING (SIZE(0 .. 255)) The XML type BoundedString shall be used in the schema for these elements.

Example:

```
<xsd:element name="onStreetName" type="boundedString"/>
```

Note: The element could also be called BoundedString255 to distinguish it from possible other similar types. If there had been other commonly used in-line type definitions, they could have been defined in a similar fashion.

6.5 **UndefinedElement Type—**Name: UndefinedElement

Definition:

```
<xsd:complexType name="UndefinedElement"/>
```

Reason: Some referenced types in SAE J2354 relating to address profiles, incomplete work, or simply unknown data structures were not defined. The type named UndefinedElement replaced the undefined types. Each occurrence represents an area where further work is required to revise SAE J2354, and corresponding changes will then be made to the Schema.