TSN.1 Server User's Manual

(For Version 5.4)

Table of Contents:

1 Overview 4
1.1 Limitations 5
1.2 System Requirements 5
1.2.1 Java Class Library 5
1.2.2 .NET DLL 5
2 Setting up the TSN.1 Server 6
2.1 Java Class Library 6
2.2 .NET DLL 6
3 Using the Java API 7
3.1 The TSNS Class 7
3.1.1 Get Version 7
3.1.2 Get License 7
3.1.3 Load a TSN.1 File 7
3.1.4 Lookup a 32-Bit Constant 7
3.1.5 Lookup a 64-Bit Constant 7
3.1.6 Lookup a String 8
3.1.7 Lookup Enumerated Literals 8
3.1.8 Lookup a Enumerated Literal 8
3.1.9 Create a Message 8
3.2 The Message Class 8
3.2.1 Access Message Fields 8
3.2.2 Remove 10
3.2.3 SizeOf 10
3.2.4 SetLength 10
3.2.5 Clone 10
3.2.6 Equals 10
3.2.7 Pack 11
3.2.8 Unpack 11
3.2.9 Print 11
3.2.10 Scan 12
3.2.11 PrintXML 12
3.2.12 ScanXML 13
3.2.13 GetBytes 13
3.2.14 Accept 13
3.2.15 Unbounded Arrays 13
3.2.16 Optional Fields 14
3.2.17 Field Conflicts 14
3.2.18 Field References and Duplicate Fields 14
3.3 Custom Message Visitors 15
3.3.1 Field 15
3.3.2 BitField 15
3.3.3 StringField 16
3.3.4 MsgField 16
3.3.5 ArrayField 17

3.3.6 Example 17
3.4 Error Handling 18
3.5 Examples 19
4 Using the .NET DLL API 22
5 Extensions 23
5.1 Display 23
5.1.1 Example 23
5.2 Endianness 24
5.2.1 Example 24
5.3 Literal 24
5.3.1 Example 25
References 26

List of Figures:

Figure 1 The Transfer Syntax Notation One Server 5

1 Overview

The TSN.1 Server is a *dynamic* message processing engine. It compiles TSN.1 message definitions at runtime. Unlike the TSN.1 Compiler, the Server does not generate any code. It stores the message definitions internally. The user applications can then manipulate these messages on the fly by using the TSN.1 Server API. This API offers similar functions to that of the TSN.1 Compiler Runtime Library [2]. These functions include:

- SizeOf – Compute the size of a message in bits
- Clone – Make a deep copy of a message
- Equal – Compare two messages for equality
- Pack – Pack a message into a bit buffer
- Unpack – Unpack a message from a bit buffer
- Print – Print a message into a string
- Scan – Parse a message from a string
- PrintXML – Print a message into a XML string
- ScanXML – Parse a message from a XML string

Typical applications using the TSN.1 Server are protocol analyzers, diagnostic monitors, and logging utilities. The Server is provided as a library with an API. It comes either as a Java class library or as a Windows .NET DLL. Therefore it runs on a diverse set of platforms and can be used from a number of programming languages. The figure below illustrates a typical usage of the TSN.1 Server:

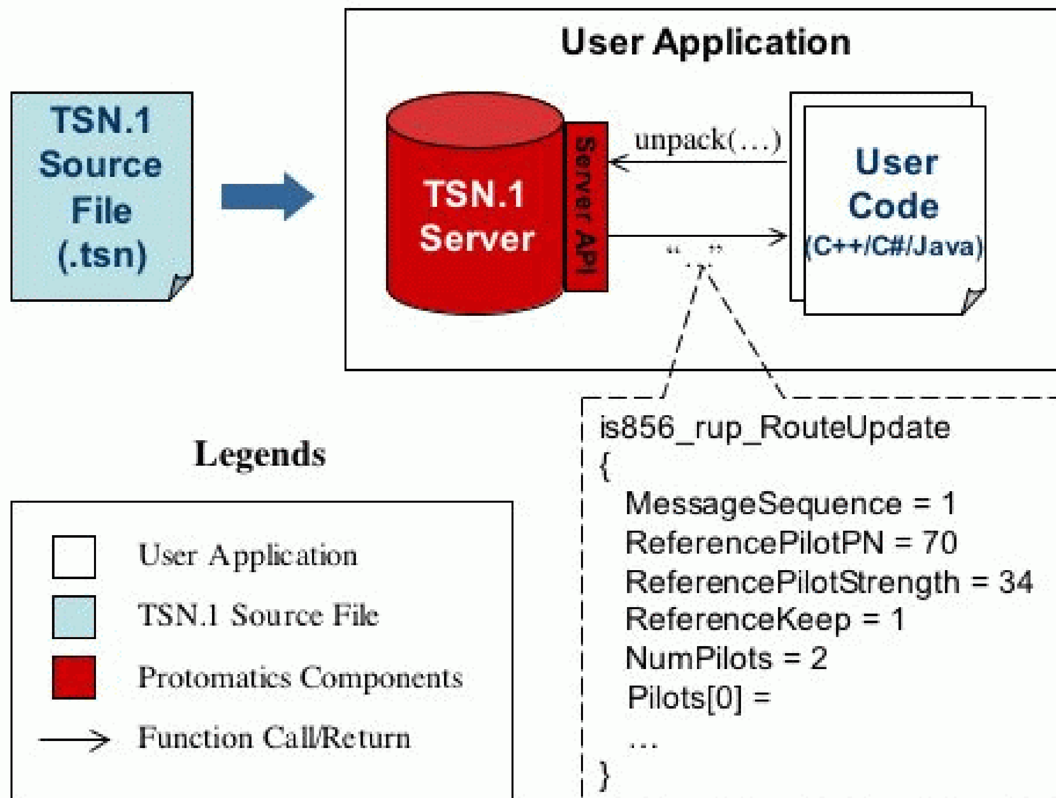

Figure 1 The Transfer Syntax Notation One Server

The TSN.1 Server is capable of parsing all TSN.1 syntax as specified by the Transfer Syntax Notation One Specification [1].

1.1 Limitations

The following are some limitations imposed by the Server. Most of them are for practical reasons.

- The maximum size of an array is limited to 2147483647.
- The maximum size of a bit field is 2147483647 bits or 256 Mbytes.
- The Server does not check for integer expression overflow.
- The Server does not check if the shift count of a left or right shift operation is non-negative or less than the width of the value being shifted.
- The Server does not check if the value of a bit field is out of range.
- When passing an argument to a message by value, the Server does not check if the value exceeds the range permitted by the formal argument. Similarly this check is also not performed when assigning values to a variable in an assignment statement.

1.2 System Requirements

1.2.1 Java Class Library

Any system supporting Java Runtime (1.5.0 or above) such as Windows. Linux, SUN Solaris, HP-UX, IBM AIX, etc

1.2.2 .NET DLL

- 32-bit Intel® Pentium® Processor 800 MHz or higher
- Microsoft® Windows 2000, Microsoft® Windows XP, or any other system supporting Microsoft® .NET Framework 1.0
- 256MB of RAM
- 10MB of available hard-disk space

2 Setting up the TSN.1 Server

2.1 Java Class Library

For Java applications, the TSN.1 Server is packaged in a single jar file, “tsns.jar”. To set up the Server, simply store the jar file on your CLASSPATH where your Java program can access. Please refer to your Java software development toolkit documentation for detailed information on how to set the CLASSPATH.

2.2 .NET DLL

For applications developed using the Microsoft .NET platform, the TSN.1 Server is delivered as a .NET DLL, “tsns.dll”. To set it up, simply store the DLL file where your application can make a reference.

3 Using the Java API

The TSN.1 Server Java API is provided through two public classes: TSNS and Message. Both classes are defined as part of the “com.protomatics.tsn.tsns” package. Import the two classes into your Java program, and use their public methods to access the Server functions.

3.1 The TSNS Class

The TSNS class provides the top-level interface for the Server. Use this class to load a TSN.1 file into the Server, lookup the value of a constant, and create a new message. You can instantiate multiple TSNS objects within your application.

3.1.1 Get Version

Use the getVersion method to look up the version the TSN.1 Server in use.

```
public static String getVersion();
```

3.1.2 Get License

Before using the Server, you must call the getLicense method. Pass the license information string to this method to obtain the license.

```
public static void getLicense(String license);
```

3.1.3 Load a TSN.1 File

Use the load method to load a TSN.1 file into the Server. The method takes two String arguments. Use the first argument to specify the compile options. Valid options are “-int64”, “-little-endian”, “-Did[=def]”, and “-sourcepath <directories>”. Use the “-D” to pass predefined constants into the TSN.1 file and use the “-sourcepath” option to specify the import directories. Multiple directories are separated by “;”. If a directory contains space characters, replace them with ‘+’ before passing them to load. The second argument to the load method specifies the path of the TSN.1 file to be loaded.

```
public void load(String options, String tsnPath);
```

3.1.4 Lookup a 32-Bit Constant

To lookup the value of a 32-bit constant defined in a TSN.1 file, use the getConstant method.

```
public int getConstant(String name);
```

3.1.5 Lookup a 64-Bit Constant

To lookup the value of a 64-bit constant defined a TSN.1 file, use the getLongConstant method.

```
public long getLongConstant(String name);
```

3.1.6 Lookup a String

To lookup the value of a string defined in a TSN.1 file, use the getString method.

```
public String getString(String name);
```

3.1.7 Lookup Enumerated Literals

To lookup the string literals of an enumerated, use the getEnumLiterals method. The method returns a "java.util.Enumeration", which enumerates all the literals in the enumerated.

```
public Enumeration getEnumLiterals(String enumName);
```

3.1.8 Lookup a Enumerated Literal

To lookup the string literal of an enumerated for a given value, use the getEnumLiteral method.

```
public String getEnumLiteral(String enumName, long value);
```

3.1.9 Create a Message

To instantiate a message defined in the TSN.1 file, use the createMessage method. If a message takes arguments, they can be set just like any bit fields in the message. See section 3.2.1.1 for details.

```
public Message createMessage(String name);
```

A message can be serialized into a sequence of bytes (3.2.13). This is for data persistence or transmission over a network. To recreate a message from a sequence of bytes, use the following overloaded createMessage method:

```
public Message createMessage(byte bytes[]);
```

For this function to behave correctly, the same message definition that was used to create the message data must be loaded into the Server.

3.2 The Message Class

The Message class represents a generic TSN.1 message. The Message class defines methods to access the message fields and perform various message operations. Always use the TSNS.createMessage interface to instantiate a message object.

3.2.1 Access Message Fields

The Message class defines a number of get and set methods to get and set the values of the message fields. The get and set methods takes a string argument "path" to specify the field being get and set. For arrays, the index is encoded as part of this string. For example, if field "A" is an array, use "A[1]" to get the second element of the array.

3.2.1.1 Bit Fields

For bit fields with a maximum size less than or equal to 32 bits, use the get and set methods to get and set their values.

```
public int get(String path);
public int set(String path, int value);
```

For bit fields with a maximum size less than or equal to 64 bits, use the getLong and setLong methods to get and set their values if the "-int64" option is specified.

```
public long getLong(String path);
public long setLong(String path, long value);
```

Bit fields are unsigned integers. However, Java does not have unsigned integer types. The "int" argument of the "set" function and the "long" argument of the "setLong" function specifies the two's complement representation of the unsigned integer to be set. Similarly, the return values of the "get" and "getLong" functions return the two's complement representation of an unsigned integer.

To get the minimum and maximum values of a bit field, use the getMin and getMax method.

```
public long getMin(String path);
public long getMax(String path);
```

3.2.1.2 Bit Buffers

For bit fields with a maximum size greater than 32 bits (64 bits if "-int64" option is specified), use the getBuf and setBuf methods to get and set their values. Use the isBuf method to test if a field is a bit buffer.

```
public boolean isBuf(String path);
public byte[]  getBuf(String path);
public void    setBuf(String path, byte buf[]);
```

3.2.1.3 Strings

To access the string fields in a message, use the getString and setString methods.

```
public int[] getString(String path);
public void  setString(String path, int str[]);
public void  setString(String path, String str);
```

The setString method is overloaded to take either an array of integers or in the case of an ASCII string, a String object. The integer array that contains the string should always be terminated by the null character.

3.2.1.4 Nested Messages

For nested message fields, use the getMsg method to get a reference of the nested message and use the setMsg method to set a nested message.

```
public Message getMsg(String path);
public Message setMsg(String path, Message msg);
```

3.2.1.5 Case Of

The access path for members of a case of field is constructed by concatenating the name of the case of field with the name of the member using the "." character. For example, "messages.RouteUpdate".

If you have a case of field with nested messages as its members and you do not know in advance which member is present, you can use the getCaseOfMsg method to retrieve the member. For example, "getCaseOfMsg("messages")".

```
public Message getCaseOfMsg(String path);
```

3.2.2 Remove

The remove method removes a field from the message. The field to be removed is specified by the argument "path", which can be an array element.

```
public void remove(String path);
```

3.2.3 SizeOf

The sizeOf method returns the size of a message in bits. This method also automatically sets the length fields in a size constraint. See SetLength (3.2.4) for details.

```
public int sizeOf();
```

3.2.4 SetLength

If a size constraint is specified for a nested message or a case of field, this method automatically sets the length field in the constraint if the length field is zero. For this to work, the size constraint must only involve simple arithmetic and the length field can only appear once, for example, "Length * 8", "(Length + 1) * 8", etc.

```
public void setLength();
```

3.2.5 Clone

The clone method makes a copy of a message.

```
public Object clone();
```

3.2.6 Equals

The equal method compares two messages for equality. The method first makes sure the two messages are of the same type, then packs the two messages into separate buffers, and finally compares the bits in the buffers for equality. It does not do a member-wise comparison.

```
public boolean equals(Object obj);
```

3.2.7 Pack

The pack method packs a message into a byte buffer. The number of bits packed is returned. The “offset” argument specifies the bit offset to start packing. The “length” argument specifies the number of available bits in the buffer. For example, if “offset” is 3, then “length” should be set to “buf.length * 8 – 3”.

```
public int pack(byte buf[], int offset, int length);
```

3.2.8 Unpack

The unpack method unpacks a message from a byte buffer. The number of bits unpacked is returned. The “offset” argument specifies the bit offset to start unpacking. The “length” argument specifies the number of available bits in the buffer. For example, if “offset” is 3, then “length” should be set to “buf.length * 8 – 3”.

```
public int unpack(byte buf[], int offset, int length);
```

3.2.9 Print

The print method prints the content of a message to a string.

```
public String print();
```

The following example illustrates the output of an IS856_rup_RouteUpdate message:

```
is856_rup_RouteUpdate
{
   MessageSequence = 2
   ReferencePilotPN = 12
   ReferencePilotStrength = 26
   ReferenceKeep = 1
   NumPilots = 2

   Pilots[0] =
   {
      PilotPNPhase = 131
      ChannelIncluded = 1
      Channel =
      {
         SystemType = 0
         BandClass  = 1
         ChannelNumber = 125
      }
      PilotStrength = 18
      Keep = 1
   }
   Pilots[1] =
   {
      PilotPNPhase = 178
      ChannelIncluded = 0
      PilotStrength = 3
      Keep = 0
   }
}
```

3.2.10 Scan

The scan method reads a string and parses it into a message. The scan method can parse strings output by the print method.

```
public void scan(String text);
```

3.2.11 PrintXML

The printXML method prints the content of a message to a XML string.

```
public String printXML();
```

The following example illustrates the output of an IS856_rup_RouteUpdate message:

```
<is856_rup_RouteUpdate>
   <MessageSequence>2</MessageSequence>
   <ReferencePilotPN>12</ReferencePilotPN>
   <ReferencePilotStrength>26</ReferencePilotStrength>
   <ReferenceKeep>1</ReferenceKeep>
   <NumPilots>2</NumPilots>
   <Pilots>
      <is856_rup_RouteUpdate_Pilots>
         <PilotPNPhase>131</PilotPNPhase>
         <ChannelIncluded>1</ChannelIncluded>
         <Channel>
            <is856_ChannelRecord>
               <SystemType>0</SystemType>
               <BandClass>1</BandClass>
               <ChannelNumber>125</ChannelNumber>
            </is856_ChannelRecord>
         </Channel>
         <PilotStrength>18</PilotStrength>
         <Keep>1</Keep>
      </is856_rup_RouteUpdate_Pilots>
      <is856_rup_RouteUpdate_Pilots>
         <PilotPNPhase>178</PilotPNPhase>
         <ChannelIncluded>0</ChannelIncluded>
         <PilotStrength>3</PilotStrength>
         <Keep>0</Keep>
      </is856_rup_RouteUpdate_Pilots>
   <Pilots>
</is856_rup_RouteUpdate>
```

3.2.12 ScanXML

The scan method reads a XML string and parses it into a message. The scanXML method can parse strings output by the printXML method.

```
public void scanXML(String text);
```

3.2.13 GetBytes

The getBytes method returns a sequence of bytes, which contains all the information necessary to recreate the message later (3.1.9).

```
public byte[] getBytes();
```

3.2.14 Accept

The accept method accepts a custom message visitor () to visit the message fields.

```
public void accept(MsgVisitor visitor);
```

3.2.15 Unbounded Arrays

Unbounded arrays in TSN.1 are denoted using a pair of empty brackets “[]”. They can only appear at the end of a message. Generally speaking, when unpacking such a

message, the size of the array cannot be determined until the data buffer is exhausted. By default, the Server reserves a fixed size array with a maximum size of 16. You can override this default maximum value by specifying an interval within the brackets, for example, "[(0 .. 32)]". The actual size of the array is indicated by a hidden bit field "_<array_name>_size_" in the message. This field is set by the user before packing a message, and set by the unpack function during unpacking.

3.2.16 Optional Fields

For optional fields, the hidden field "_<optional_name>_optional_" indicates if the optional fields are present. This field is set by the user before packing a message, and set by the unpack function during unpacking.

3.2.17 Field Conflicts

Fields with the same name may appear in different branches of a conditional without causing any conflict. In this case, the Server represents the field using the *maximum* compatible type for the field. For example, if a field is mapped to a "int" in one branch, and a "long" in another, then "long" is used. If no maximum compatible type is possible, the Server reports an error.

3.2.18 Field References and Duplicate Fields

Semantic check for field references and duplicate fields can be problematic in TSN.1 when fields are defined inside conditionals. The following example illustrates the challenge.

```
FieldReference() ::=
{
    A  8;

    if(A == 3)
    {
       B  8;
    }

    if(A > 2 && A < 4)
    {
       C  B * 8;
    }
}
```

In this example, field "B" is not always present in the message. The Server, therefore, needs to check that any subsequent reference to "B" in an expression only happens when "B" is present. The reference to "B" in the expression "B * 8" is obviously valid in this example since the guard condition "A > 2 && A < 4" implies "A == 3". However, this is not always computable at load time because the guard condition can be arbitrarily complex. When such computation is not possible, the Server performs the check at run-time. The check for duplicate fields is similarly handled.

3.3 Custom Message Visitors

You can implement custom message visitors to iterate over the message fields. This is accomplished by inheriting from the abstract class MsgVisitor:

```
public abstract class MsgVisitor
{
   public void visit(BitField field)
   {
   }

   public void visit(StringField field)
   {
   }

   public void visit(MsgField field)
   {
   }

   public void visit(ArrayField field)
   {
   }
}
```

The BitField, MsgField, and ArrayField classes represent bit fields, nested message fields, and arrays respectively. They all derive from the common base class Field. The following sub-sections describe the interface of these classes.

3.3.1 Field

The abstract base class for the fields in a message.

3.3.1.1 GetName

The getName method returns the name of the field.

```
public String getName();
```

3.3.1.2 GetCanonicalName

The getCanonicalName method works just like the GetName method except for case of field members it returns the name of the case of field concatenated with the name of the field with a the “.” character in between.

3.3.1.3 Accept

The accept method is an abstract method that allows a message visitor to visit the field.

```
public abstract void accept(MsgVisitor visitor);
```

3.3.2 BitField

A bit field can be an int (<= 32 bit), a long (> 32 bit && <= 64 bit), or a bit buffer (> 64 bit).

3.3.2.1 IsLong

The isLong method returns true if the bit field is a long.

```
public boolean isLong();
```

3.3.2.2 IsBuf

The isBuf method returns true if the bit field is a bit buffer.

```
public boolean isBuf();
```

3.3.2.3 Get

The get method returns the int value of the bit field. Use this method only if the bit field is an int.

```
public int get();
```

3.3.2.4 GetLong

The get method returns the long value of the bit field. Use this method only if the bit field is a long.

```
public long getLong();
```

3.3.2.5 GetBuf

The get method returns the buffer of the bit field. Use this method only if the bit field is a bit buffer.

```
public byte[] getBuf();
```

3.3.2.6 GetValueString

The getValueString method returns the value of the bit field in a string.

```
public String getValueString();
```

3.3.3 StringField

The StringField class represents the string fields in a message.

3.3.3.1 Get

The get method returns the int array that contains the characters if the string.

```
public int[] get();
```

3.3.3.2 GetValueString

The getValueString method returns the value of the string field in a string.

```
public String getValueString();
```

3.3.4 MsgField

The MsgField class represents the nested messages in a message.

3.3.4.1 GetMsg

The getMsg method returns the nested message.

```
public Message getMsg();
```

3.3.5 ArrayField

The ArrayField class represents the arrays in a message.

3.3.5.1 GetSize

The getSize method returns the size of the array.

```
public int getSize();
```

3.3.5.2 Get

The get method returns the array element at the specified index.

```
public Field get(int index);
```

3.3.6 Example

The following example illustrates how to implement a custom message visitor that prints a message to a XML string. To visit a message, create an instance of the visitor and pass it to the accept method of the Message class.

```
class PrintXMLMsgVisitor extends MsgVisitor
{
   public void visit(BitField field)
   {
      String name = field.getName();

      text.append("<");
      text.append(name);
      text.append(">");
      text.append(field.getValueString());
      text.append("</");
      text.append(name);
      text.append(">");
   }

   public void visit(StringField field)
   {
      String name = field.getName();

      text.append("<");
      text.append(name);
      text.append(">");
      text.append(field.getValueString());
      text.append("</");
      text.append(name);
      text.append(">");
   }

   public void visit(MsgField field)
   {
      String name = field.getName();

      text.append("<");
      text.append(name);
      text.append(">");
      field.getMsg().accept(this);
      text.append("</");
      text.append(name);
      text.append(">");
   }

   public void visit(ArrayField field)
   {
      for(int i = 0; i < field.getSize(); ++i)
      {
         field.get(i).accept(this);
      }
   }

   StringBuffer text = new StringBuffer();
}
```

3.4 Error Handling

When the Server detects an error, a generic Java Exception is thrown. Your program can catch this exception and use the getMessage method of the Exception to get the error string. Alternatively, you can also use the getErrorString method to get the error string. Both the TSNS and the Message class implement this method.

3.5 Examples

The example in this section uses the following TSN.1 definition:

```
...

is856_rup_RouteUpdate() ::=
{
   MessageSequence          8;
   ReferencePilotPN         9;
   ReferencePilotStrength   6;
   ReferenceKeep            1;
   NumPilots                4;

   Pilots[NumPilots]  :
   {
      PilotPNPhase     15;
      ChannelIncluded  1;

      if(ChannelIncluded == 1)
      {
         Channel  : is856_ChannelRecord;
      }

      PilotStrength  6;
      Keep           1;
   }

   align(8);
}
```

The following Java code snippet illustrates the usage of some of the TSN.1 Server Java API.

```
TSNS    tsns = new TSNS();
Message rup, rup2;

try
{
   TSNS.getLicense("Your License Information");

   tsns.load
      (
         "-sourcepath /project/is856/src;.",
         "/project/is856/src/is856_rup.tsn"
      );

   rup  = tsns.createMessage("is856_rup_RouteUpdate");
   rup2 = tsns.createMessage("is856_rup_RouteUpdate");
}
catch(Exception e)
{
   System.err.println(tsns.getErrorString());
   System.exit(1);
}

try
{
   rup.set("MessageSequence", 1);
   rup.set("ReferencePilotPN", 12);
   rup.set("ReferencePilotStrength", 27);
   rup.set("ReferenceKeep", 1);
   rup.set("NumPilots", 2);

   rup.getMsg("Pilots[0]").set("PilotPNPhase", 0x56);
   rup.getMsg("Pilots[0]").set("ChannelIncluded", 0);
   rup.getMsg("Pilots[0]").set("PilotStrength", 20);
   rup.getMsg("Pilots[0]").set("Keep", 1);

   rup.getMsg("Pilots[1]").set("PilotPNPhase", 0x3e);
   rup.getMsg("Pilots[1]").set("ChannelIncluded", 1);

   Message record = rup.getMsg("Pilots[1]").getMsg("Channel");
   // populate the record.

   rup.getMsg("Pilots[1]").set("PilotStrength", 7);
   rup.getMsg("Pilots[1]").set("Keep", 0);

   int numBit = rup.sizeOf();

   byte buf[] = new byte[(numBit + 7) / 8];

   int numBitPacked = rup.pack(buf, 0, buf.length * 8);

   int numBitUnpacked = rup2.unpack(buf, 0, numBit);

   if(!rup.equals(rup2))
   {
      System.err.println("rup and rup2 are not equal.");
   }
```

```
        String text = rup.toString();

        rup2.scan(text);

        if(!rup.equals(rup2))
        {
           System.err.println("rup and rup2 are not equal.");
        }
     }
     catch(Exception e)
     {
        System.err.println(rup.getErrorString());
        System.err.println(rup2.getErrorString());
        System.exit(1);
     }
```

4 Using the .NET DLL API

The usage of the .NET DLL API is essentially the same as that of the Java API. Import the .NET DLL (“tsns.dll”) assembly into your application. The TSNS and Message classes are defined under the namespace “com.protomatics.tsn.tsns”. With the .NET platform, it is very easy to access the API from any of the .NET programming languages such as Visual Basic, C++, C#, and J#. Please refer to Chapter 3 for details of the API.

5 Extensions

Extensions are XML [3] markups in a TSN.1 file. They are designed to provide directives or hints to the Server as well as other TSN.1 development tools. For example, the display extension (5.1) is used to indicate the format in which a bit field is to be displayed. Extensions are not part of the TSN.1 notation.

Each extension in the TSN.1 file is specified as an XML element. Because extensions are XML elements, the usual XML escape characters apply, for example, “<” for “<”, “>” for “>”, and “&” for “&”.

The subsequent sections of this chapter describe the extensions supported by the Server.

5.1 Display

The display extension is used to specify the format in which bit fields are to be displayed. The extension can appear either at the top level or right after a bit field declaration. The valid values are: “oct” for octal, “dec” for decimal, “hex” for hexadecimal, “ipv4” for IPv4 address, “ipv6” for IPv6 address, “ascii” for ASCII strings, and “none” to disable the display of the field. The default value is “dec”. There is an optional “def” attribute. When present, all fields in the message will be displayed in the specified format. If the “def” attribute is not present, all messages defined after the display extension will use the specified display. Subsequent display extension in the same file overrides the then current display extension starting from the point where the subsequent display extension is specified.

The display extension has an optional “value_offset” attribute. The value of this attribute is an integer. In certain type of encoding, for example ASN.1 PER [4], an integer value is represented at an offset to its two’s complement representation. In these cases, the “value_offset” attribute can be used to specify the offset value to be added to the decoded value before display.

5.1.1 Example

```
DisplayExample() ::=
{
   TransactionID  8; <display>hex</display>

   StringLength  8;
   String        8 * Length; <display>ascii</display>
}

DisplayExample2() ::=
{
   TransactionID  8;

   // The range of values is shifted by -5, from
   // [0 - 255] to [-5, 250]
   Value  8; <display value_offset=”-5”/>
}

<display def=”DisplayExample2”>hex</display>
```

5.2 Endianness

TSN.1 specifies messages be encoded in the big endian format. You can override this default by using the endianness extension. The endianness extension can appear either at the top level or right after a bit field declaration. The valid values for the extension are “big” and “little”. When the extension appears at the top level of a TSN.1 file, the attribute “def” must be specified. When the attribute value is a valid message name, the specified endianness applies to the entire message.

5.2.1 Example

```
EndiannessExample() ::=
{
   Value   16;  <endianness>little<endianness>
}

EndiannessExample2() ::=
{
   Value  32;
}

<endianness def=”EndiannessExample2”>little</endianness>
```

5.3 Literal

The literal extension is used to associate an arbitrary string to the enum literal for display. It overrides the default string, which is the enum literal itself.

5.3.1 Example

```
is856_admp_AirMessageId ::= enumerated
{
   IS856_ADMP_UATI_REQUEST,    <literal>UATI Request</literal>
   IS856_ADMP_UATI_ASSIGNMENT, <literal>UATI Assignment</literal>
   IS856_ADMP_UATI_COMPLETE,   <literal>UATI Complete</literal>
   ...
}
```

References

[1] The Transfer Syntax Notation One Specification
[2] The TSN.1 Compiler User’s Manual
[3] Extensible Markup Language (XML) 1.0, W3C Recommendation
[4] ASN.1 encoding rules: Specification of Packed Encoding Rules (PER), ITU-T Recommendation X.691, International Standard 8825-2