Alternative binary representations of the XML Information Set based on ASN.1

OSS Nokalva, Inc.

August 11, 2003

Abstract

Two recent standard initiatives involving the use of ASN.1 in support of XML and Web Services are being pursued by a joint committee of ISO/IEC and ITU-T. One of the standards (X.694) specifies a mapping from XML Schema to ASN.1. Another (X.695) provides support for fast web services and specifies, among other things, an ASN.1 Schema for the XML Information Set. Performance tests have been conducted on binary encoding/decoding of instances whose schema was translated from XML Schema to ASN.1 using X.694. Other performance tests have been conducted on the ASN.1 Schema for the XML Information Set. The results of these tests are presented in this paper.

CONTENTS

1. Introduction

Two recent standard initiatives involving the use of ASN.1 in support of XML and Web Services are being pursued by a joint committee of ISO/IEC and ITU-T.

One of them has resulted in the specification of a new set of encoding rules for ASN.1, the "Extended XML Encoding Rules" (X.693 Amendment 1) and in the specification of a mapping from XML Schema to ASN.1 (X.694). A representation of the infoset based on X.694 is described in Section 2, together with the results of a performance test.

The other addresses Web Services and the XML Information Set and aims to provide efficient alternatives to the use of XML 1.0. A representation of the infoset based on the ASN.1 Schema for the Information Set is described in Section 3, together with the results of a performance test.

2. Binary representation of XML documents using X.694

2.1 X.694

The joint committee of the ISO/IEC and ITU-T in charge of the ASN.1 standards has produced a draft standard that specifies a mapping from W3C XML Schema to ASN.1, which is currently under Final Committee Draft ballot in ISO. This work is expected to produce a new Recommendation | International Standard named ITU-T Rec. X.694 | ISO/IEC 8825-5 - Mapping W3C XML Schema Definitions into ASN.1.

The goal of this standard is to enable applications to use the binary encoding rules of ASN.1 (such as the Basic Encoding Rules - BER - or the Packed Encoding Rules - PER) for efficient transfer of instances conforming to a schema written in W3C XML Schema. This is achieved by first converting an XML schema into an ASN.1 schema and then encoding/decoding instances using PER or BER.

Applications written using X.694 will benefit from a compact serialization format and from high data encoding/decoding speed when using the Packed Encoding Rules (PER). In addition,

applications will be able to exchange instances serialized in XML 1.0 by using the Extended XML Encoding Rules of ASN.1 (EXTENDED-XER).

2.2 Performance test

A test was conducted at OSS Nokalva to compare the speed and size of a binary representation of XML based on X.694 against several other alternatives. For compression testing, PER and zlib were used. For speed testing, comparisons were done between the OSS XSD/C Tools and Xerces 2.3.0 SAX and DOM APIs.

Several XML instances varying in size were used in the test.

The following tools were used in the test:

OSS XSD/C Tools (1.0 betaA) (www.oss.com)

Zlib 1.1.4 (www.gzip.org/zlib)
The highest compression level (9) was chosen.

Xerces 2.3.0
SAX/DOM APIs

2.2.1 Compression test

This test was performed on several real-life XML instances publicly available on the Web. UDDI[1] schemas and instances, UDS[2] schemas and instances, UBL[3] schemas and instances, and other schemas and instances were used in the test. Multiple instances were processed for each schema. Results are shown as averages over all the instances relative to each schema.

2.2.2 Speed test

This test was performed to compare parsing/writing speeds between the OSS XSD tools and the Xerces 2.3.0 SAX and DOM APIs. One purpose of this test was to measure the speed difference between parsing binary encodings and parsing XML documents. Another purpose was to measure the speed difference between creating XML documents and producing binary encodings. Encoding and decoding times were measured as follows:

OSS XSD tools

[1] Universal Description, Discovery and Integration of Web Services (http://www.uddi.org/)
[2] Unified Directory Specification (ITU-T Rec. F.515)
[3] Universal Business Language (http://oasis-open.org/committees/ubl/)

Encoding time is the time spent converting the data from in-memory data structures to binary encodings.

Decoding time is the time spent decoding binary encodings and populating in-memory data structures.

Xerces SAX APIs

Encoding time is the time spent creating an XML document from a list of buffered SAX events stored in memory and then compressing the XML document using zlib.

Decoding time is the time spent decompressing a compressed XML document using zlib and then parsing the XML document storing a list of buffered SAX events in memory.

Note: Other measurements for SAX were taken without involving the compression and decompression steps in between. Other measurements were taken by using empty event handlers for SAX events (with no program code) to store a list of buffered SAX events in memory.

Xerces DOM APIs

Encoding time is the time spent creating an XML document from a DOM tree and then compressing the XML document using zlib.

Decoding time is the time spent decompressing a compressed XML document using zlib and then parsing the decompressed XML document using the DOM parser.

Note: Other measurements for DOM were taken without involving compression and decompression steps in between.

2.2.3 Results of the tests

Legend:

- Runtime - a compression tool
- UncompSize - average size of the original XML instances for a given test (same for all compression tools)
- CompSize - average size of the compressed instances
- ExeSize - size of the standalone compression executable file
- CompRatio - compression percentage (lower is better)
- EncodeTime - average time of compression (XML creation for DOM/SAX)
- DecodeTime - average time of decompression (XML parsing for DOM/SAX)
- TotalTime - the sum of EncodeTime and DecodeTime
- eSAX - a SAX parser with empty event handlers.

Sizes are in bytes and times are in microseconds.

a. RANAP, UncompSize = 1546

Runtime	CompSize	CompRatio	EncodeTime	DecodeTime	TotalTime	ExeSize
OSS PER	26	1.70%	15.7	14.6	30.3	684832
OSS PER+Zlib	35	2.30%	367.2	27.3	394.5	721412
Zlib	411	26.60%	475.8	49.4	525.2	435760
Xerces DOM	-	-	154.4	380.0	534.4	2800224
Xerces SAX	-	-	90.6	284.4	375.0	2823032
Xerces eSAX	-	-	-	235.8	-	2819896
Xerces DOM+Zlib	398	25.70%	614.4	432.4	1046.8	2838668
Xerces SAX+Zlib	405	26.20%	536.0	337.6	873.6	2861668
Xerces eSAX+Zlib	411	26.60%	-	272.4	-	2857028

b. UBL, UncompSize = 36769, averaged on 11 instances

UBL, UncompSize = 36769, averaged on 11 instances

Runtime	CompSize	CompRatio	EncodeTime	DecodeTime	TotalTime	ExeSize
OSS PER	4426	12.00%	944.5	1777.5	2722.0	858500
OSS PER+Zlib	582	1.60%	1555.1	1984.7	3539.8	896168
Zlib	1607	4.40%	2088.5	441.0	2529.5	435856
Xerces DOM	-	-	3716.8	7635.5	11352.3	2800352
Xerces SAX	-	-	2125.5	4623.0	6748.5	2823096
Xerces eSAX	-	-	-	3422.0	-	2819992
Xerces DOM+Zlib	1603	4.40%	5857.7	8075.5	13933.2	2842028
Xerces SAX+Zlib	1565	4.30%	4098.2	5324.1	9422.3	2861732

c. UDDI, UncompSize = 2422, averaged on 13 instances

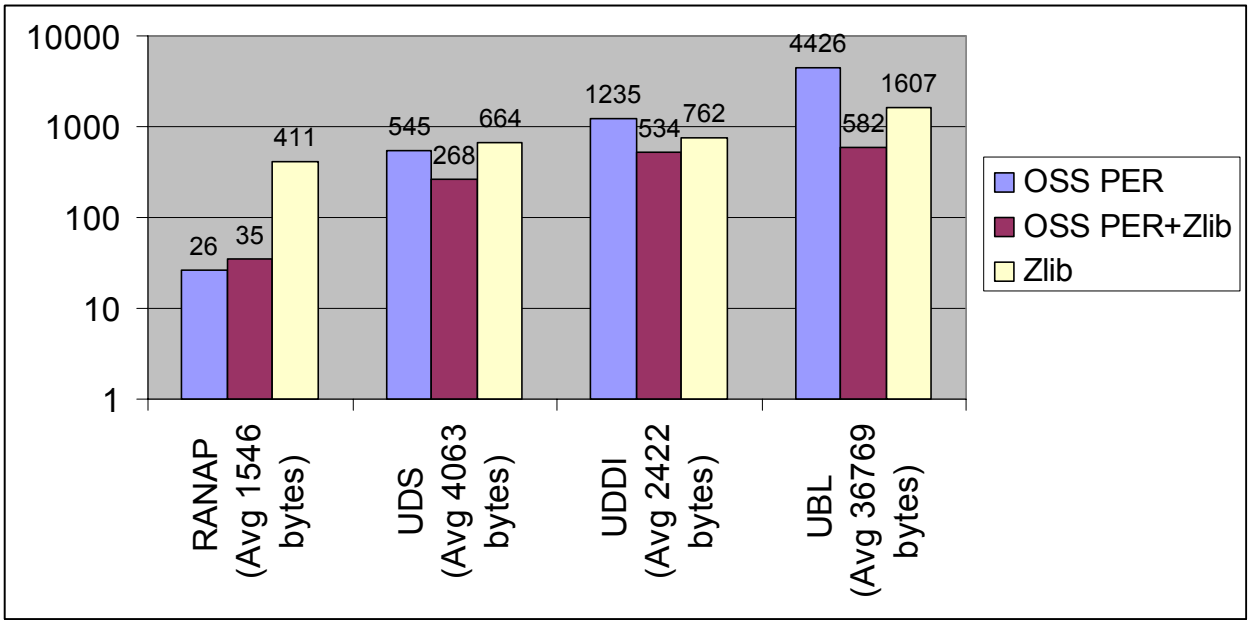
Runtime	CompSize	CompRatio	EncodeTime	DecodeTime	TotalTime	ExeSize
OSS PER	1235	51.00%	25.5	42.1	67.6	805048
OSS PER+Zlib	534	22.10%	512.9	95.5	608.4	845404
Zlib	762	31.50%	561.5	66.8	628.3	439376
Xerces DOM	-	-	187.2	406.8	593.9	2800352
Xerces SAX	-	-	113.2	306.5	419.7	2823096
Xerces eSAX	-	-	-	238.7	-	2820024
Xerces DOM+Zlib	755	31.20%	758.5	476.9	1235.4	2842028
Xerces SAX+Zlib	758	31.30%	657.2	386.9	1044.1	2861732
Xerces eSAX+Zlib	762	31.50%	-	319.0	-	2857156

d. UDS, UncompSize = 4063, averaged on 12 instances

Runtime	CompSize	CompRatio	EncodeTime	DecodeTime	TotalTime	ExeSize
OSS PER	545	13.40%	86.5	116.2	202.7	801140
OSS PER+Zlib	268	6.60%	497.7	154.5	652.2	839416
Zlib	664	16.30%	643.3	81.6	724.9	435888
Xerces DOM	-	-	427.2	915.8	1343.0	2800352
Xerces SAX	-	-	242.2	593.8	836.0	2823096
Xerces eSAX	-	-	-	433.3	-	2820024
Xerces DOM+Zlib	663	16.30%	1095.7	1012.0	2107.7	2838796
Xerces SAX+Zlib	637	15.70%	864.2	676.1	1540.3	2861732
Xerces eSAX+Zlib	664	16.30%	-	546.6	-	2857156

2.2.4 Discussion of the results

2.2.4.1 Compression results



The schemas for RANAP and UDS contain types that are heavily constrained (bounded integers, enumerations of strings and enumerations of integers). Those constrained types allow PER to achieve a better compression rate than Zlib's variant of LZ77 algorithm.

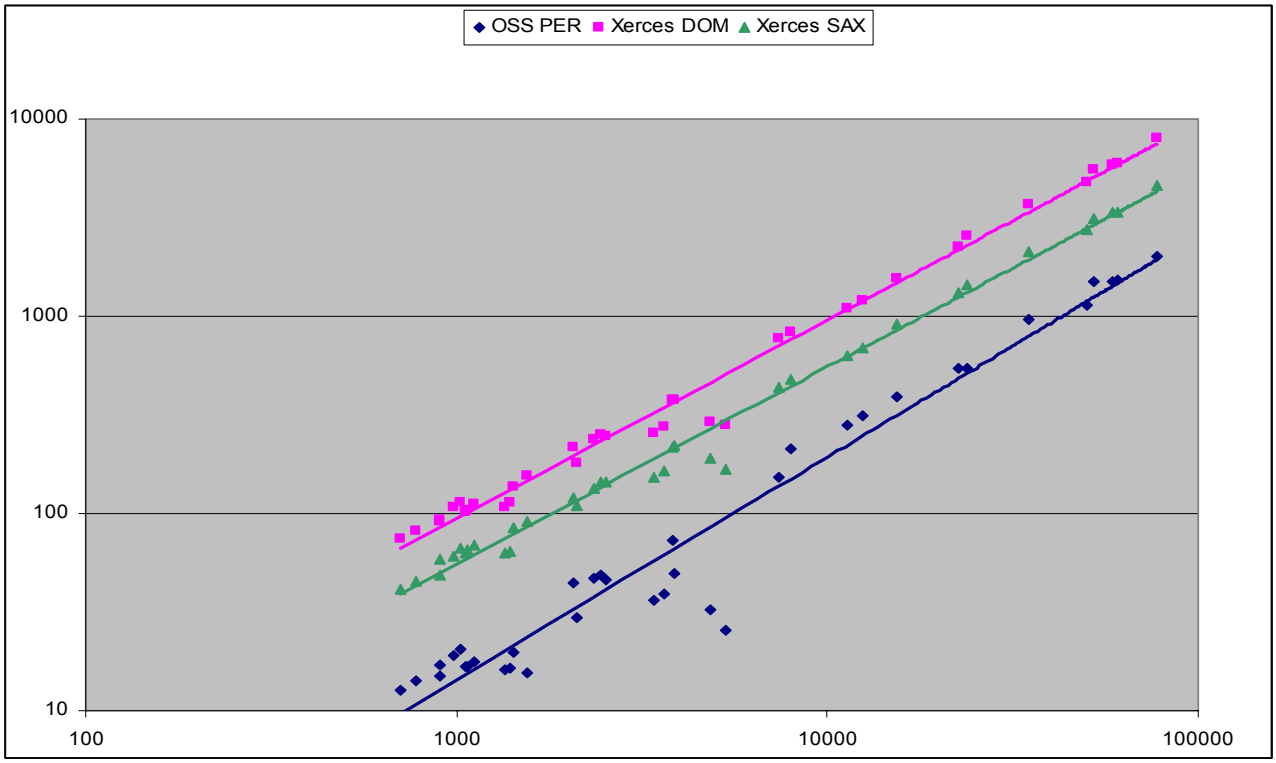
The schemas for UDDI and UBL contain types that have very weak constraints (unconstrained integers, unconstrained character strings). For these schemas, Zlib's compression algorithm achieves better compression than PER.

For the schemas that are added "optimum" constraints (RANAP), compressing the PER-encoded message with Zlib does not reduce its size. For the schemas that contain unconstrained types, compressing the PER-encoded message with Zlib allows the compression rate to be better than that obtained using PER alone.

In all the tests performed, PER+Zlib gives better results than Zlib alone. Using PER and Zlib together allows getting the best from each of the two methods. The constrained types exhibit very good compression when encoded by PER, and Zlib later performs well on the portions of the PER-encoded message corresponding to the unconstrained types.

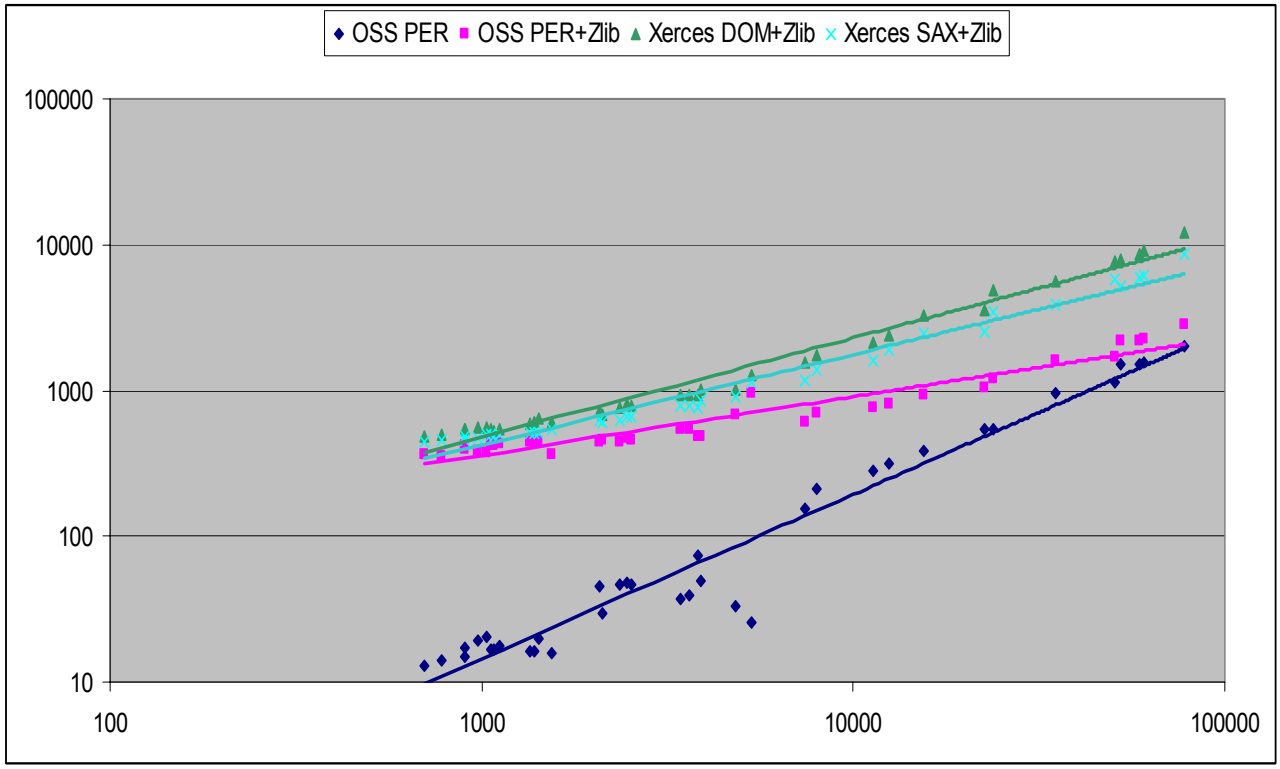
2.2.4.2 Time results

a. Encoding time without compression



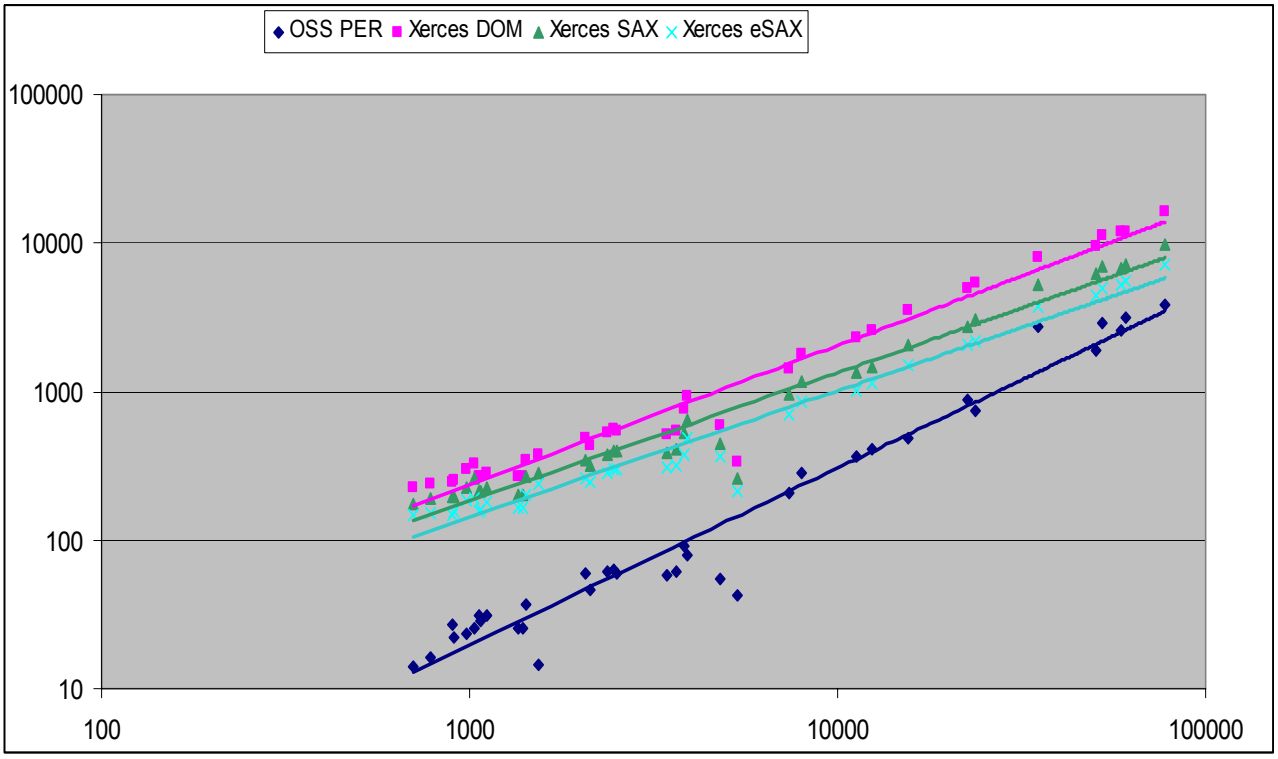
This graph (in logarithmic scale) shows that using the OSS PER encoder for building a PER message from in-memory data structures is more efficient than using DOM or SAX for building an XML message from the same in-memory data structures.

b. Encoding time with compression



This graph shows that using the OSS PER encoder for building a PER message from in-memory data structures and then compressing the PER-encoded message with Zlib is faster than building an XML message from the same in-memory data structures using DOM or SAX and compressing it.

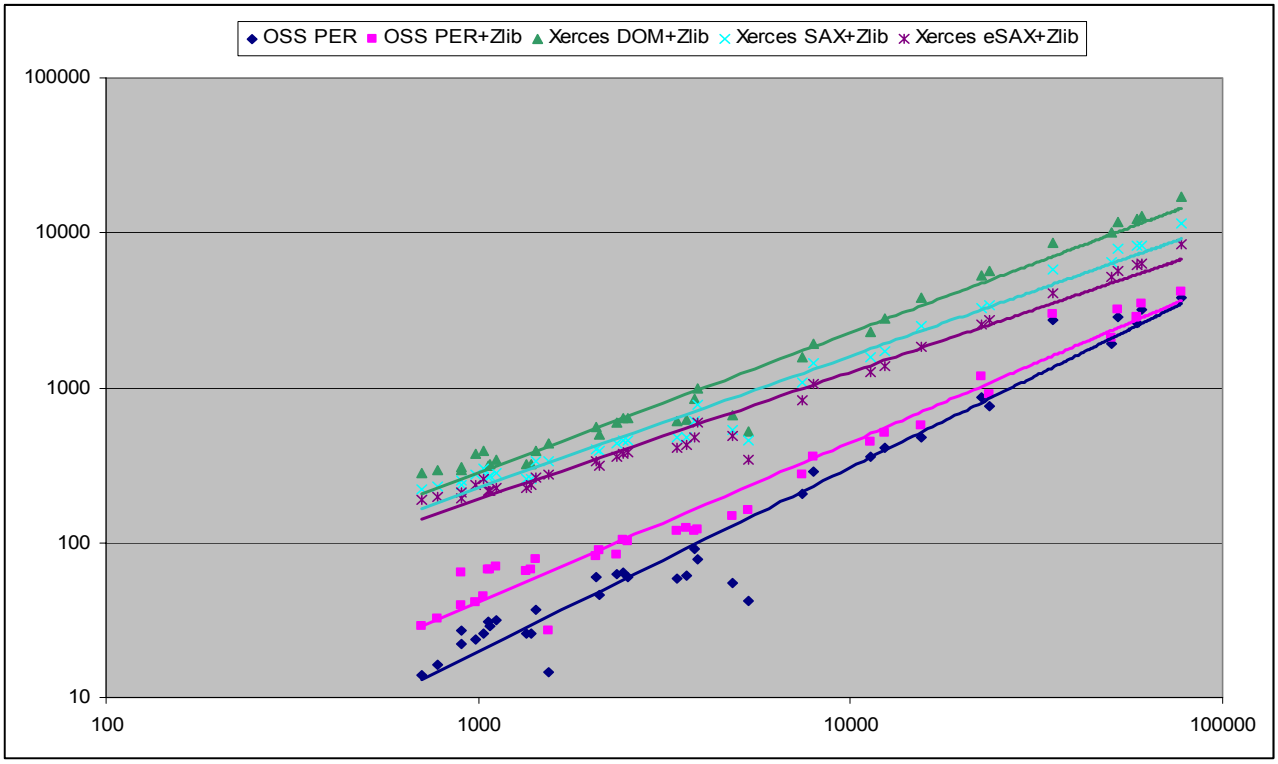
c. Decoding time without compression



This graph shows that building in-memory data structures from a PER message using the OSS PER decoder is faster than using DOM or SAX for building the same in-memory structures from an XML message.

The graph also shows that the time required to parse an XML instance using SAX with empty event handlers (no data structures built) is longer than the time required for the whole decoding process (parsing plus building in-memory data structures) using the OSS PER decoder.

d. Decoding time with compression



This graph shows that decompressing a compressed PER-encoded message using Zlib and building in-memory data structures from the PER-encoded message using the OSS PER decoder is faster than decompressing a compressed XML message and building in-memory data structures (using DOM or SAX) from the XML message.

3. Binary representation of XML documents using the ASN.1 Schema for the XML Information Set

3.1 X.695

A new standard initiative has been started recently by a joint committee of ISO/IEC and ITU-T, involving the use of ASN.1 in support of Web Services and the XML Information Set. The work is expected to produce a new Recommendation | International Standard named ITU-T Rec. X.695 | ISO/IEC 8825-6 - ASN.1 Encoding Rules - Fast Web Services.

The Web Services part of this standard will specify a set of ASN.1 type definitions (an ASN.1

schema) that are semantically equivalent to a SOAP 1.2 envelope, allowing fast binary communications as an alternative to XML 1.0-based exchanges.

The infoset part of this standard will specify an ASN.1 schema for the XML infoset, which can represent any infoset (as specified in the W3C XML Information Set Recommendation [1]) as an ASN.1 message that can be encoded in a standardized binary form.

OSS Nokalva, Inc. has developed a draft ASN.1 Schema for XML Information Set and has submitted it to the committee for inclusion in the draft standard draft. This ASN.1 schema specifies a set of type definitions for serializing a generic XML Infoset using ASN.1 binary encoding rules.

This section presents the results of a performance test that has been performed on the ASN.1 schema.

The main goal of the ASN.1 Schema for the XML Information Set is to provide a compact serialization of any XML Information Set for which no schema definition is available, without increasing parsing time or writing time. It is particularly suitable to applications for which the classic compression algorithms are too costly in terms of resources (CPU time and memory).

Internationalization and accessibility are implicitly guaranteed by directly representing information items and their properties as ASN.1 type definitions.

The prototype implementation described here requires that the schemas are entirely parsed to in-memory data structures; however other implementation can use different strategies (streaming, partial serialization, partial decoding, etc.)

The size of the XML documents used for the tests range form ~40 to ~200 Kbytes. This should not be understood as a limitation in the size of the infosets that can be handled.

Section 3.1 describes the ASN.1 schema that is part of the current draft of the standard. Section 3.2 describes the test program that has been written for performing the tests. Section 3.3 presents the results of the performance tests.

3.2 ASN.1 Schema for the XML Information Set

3.2.1 About ASN.1

Abstract Syntax Notation One (ASN.1) is a formal language for abstractly describing messages to be exchanged between distributed computer systems. ASN.1 is used in many national and international standards, including:

- X.500 Standards - The Directory;
- cellular telephony;
- security;

- videoconferencing;
- air transportation;
- messaging
- biometrics
- RSA Public-Key Cryptography Standards;

For a more complete list of standards in which ASN.1 is used, refer to http://asn1.elibel.tm.fr/uses/index.html.

ASN.1 tools (both professional tools and free tools) are available for nearly all platforms and for all of the major programming languages.

The ASN.1 language and its Encoding Rules are themselves defined by a set of international standards. These are:

- **ITU-T Rec. X.680 | ISO/IEC 8824-1** - Abstract Syntax Notation One (ASN.1) - Specification of basic notation [*2*];
- **ITU-T Rec. X.681 | ISO/IEC 8824-2** - Abstract Syntax Notation One (ASN.1) - Information Object Specification [*3*];
- **ITU-T Rec. X.682 | ISO/IEC 8824-3** - Abstract Syntax Notation One (ASN.1) - Constraint Specification [*4*];
- **ITU-T Rec. X.683 | ISO/IEC 8824-4** - Abstract Syntax Notation One (ASN.1) - Parameterization of ASN.1 Specifications [*5*];
- **ITU-T Rec. X.690 | ISO/IEC 8825-1** ASN.1 Encoding Rules - Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) [*6*];
- **ITU-T Rec. X.691 | ISO/IEC 8825-2** ASN.1 Encoding Rules - Specification of Packed Encoding Rules (PER) [*7*];
- **ITU-T Rec. X.692 | ISO/IEC 8825-3** ASN.1 Encoding Rules - Specification of Encoding Control Notation (ECN) [*8*];
- **ITU-T Rec. X.693 | ISO/IEC 8825-4** ASN.1 Encoding Rules - XML Encoding Rules[*9*].

3.2.2 Description of the ASN.1 Schema for the XML Information Set

The ASN.1 schema for the XML Information Set provides a complete and faithful abstract representation of all the information items in the XML Information Set and all their properties, as specified in the W3C XML Information Set Recommendation.

The standard ASN.1 binary encoding rules (PER, BER, etc.) provide compact serialization formats for the types defined in this ASN.1 schema, facilitating the exchange of XML infosets between systems, with significant performance benefits over the exchange of the corresponding XML documents. Unlike other existing alternative binary representations of XML 1.0, the ASN.1 schema is built upon the W3C standard specification of the XML Information Set. This allows the direct use of in-memory infosets that are never serialized into/from XML 1.0 documents.

The ASN.1 schema for the XML infoset differs from other existing alternative binary representation formats of XML conceptually in several ways:

- The ASN.1 schema is a formal specification that uses a standard data-definition language with standardized on-the-wire representations of the data. Among the benefits are: readability; clarity and freedom from ambiguity in the format specification; availability of tools (including most free ASN.1 tools) that facilitate the development of applications based on this specification.

- The ASN.1 schema has several features that an application can use to achieve various degrees of compactness, according to availability of memory, CPU cycles, and other system resources. Compression is an optional feature and operates on character-string blocks to maximize efficiency.

- Any application conceptually based on the standard XML Information Set can easily be adapted to use the ASN.1 schema. There is no loss of information when going from an application's internal representation of the abstract infoset to the ASN.1 representation and then back to the infoset, because the individual properties of information items are completely mapped into the ASN.1 schema.

3.3. Description of the test program

A program has been created in order to perform tests on the ASN.1 schema for the infoset. This program is a native Win32 DLL which uses:

- the Xerces-C++ SAX2 parser (version 2.2.0) to parse input XML documents;
- OSS Nokalva's ASN.1 tools for C to encode/decode abstract values of the ASN.1 infoset schema;
- the bzip2 compression library (version 1.0.2) to compress/decompress the component-strings table (see Annex A).

The program is able to create an in-memory representation of the XML infoset (an "infoset tree") from any of the following sources:

- an XML document (by processing Xerces' SAX event callbacks);
- a PER encoding of an abstract value of the ASN.1 infoset schema (by processing the data structures populated by the OSS PER decoder);
- another infoset tree (by traversing the source infoset tree).

Multiple infoset trees can be present in memory at the same time. The program is able to write the contents of an infoset tree to:

- an XML 1.0 document (directly, without using Xerces);
- a PER encoding of an abstract value of the ASN.1 infoset schema (by invoking the OSS PER encoder);
- another infoset tree.

The in-memory infoset representation is built around two hash tables:

- a string hash table;
- a "name" hash table.

In the present context, a "name" consists of: a namespace name (optional), a local name, and a namespace prefix (optional). The string hash table has two main purposes:

- reduce (by avoiding the storage of multiple identical strings) both the amount of virtual memory needed to host an infoset tree and the number of memory allocations performed while creating the infoset tree;
- prepare the component-strings table (see Annex A) to be included in the abstract value of the ASN.1 infoset schema, avoiding the inclusion of multiple identical strings in this table.

The main purpose of the name hash table is to prepare the common-names table (see Annex A) to be included in the abstract value of the ASN.1 infoset schema, avoiding the inclusion of multiple identical element or attribute names in this table. In the current version of the test program, there are separate string hash tables and separate name hash tables for each infoset. If two or more infoset trees are present in memory at the same time, each one has its own hash tables. (It would be useful to investigate whether any further significant speed improvements can be achieved by sharing strings among multiple infoset trees in memory.) The test program exploits many features of the ASN.1 infoset schema in order to achieve maximum compactness. In particular, it:

- gathers all the existing strings in the infoset into a single component-strings table; this includes, in particular, the strings that are namespace prefixes, local names, namespace names, attribute normalized values, and element-content character chunks;
- rearranges these strings according to their frequency in the infoset, so that the 2047 most frequent strings occupy the first 2047 positions in the component-strings table; this will result in a more compact PER encoding of the infoset (see the definition of LargeIndex in Annex A);
- always uses indexes to reference strings and never includes any strings anywhere within the abstract value except in the component-strings table;

- gathers all the existing element and attribute names in the infoset into the common-names table;
- rearranges these names according to their frequency in the infoset, so that the 15 most frequent names occupy the first 15 positions in the common-names table; this will result in a more compact PER encoding of the infoset (see the definition of SmallIndex in Annex A);
- always uses indexes to reference names and never includes any names anywhere within the abstract value except in the common-names table;
- uses the element-content-characters alternative for the element children whenever possible (that is, whenever the only child of an element is a character information item chunk which does not consist of ignorable whitespace).

The test program does not support all the features of the ASN.1 schema, but only a subset that has been considered relevant to the test. In particular, the current version of the program ignores all processing instructions, comments, document type declaration, notations, unparsed entities, base URI, character encoding scheme, version, attribute type, and (attribute) references.

For the base-URI, the same-as-parent-element choice alternative is always selected. For the in-scope-namespaces, the same-as-parent-element choice alternative is always selected. The ASN.1 schema should be improved to allow an optimized representation to be used whenever the infoset agrees with the "Namespaces in XML" scoping rules. In this case, there is much information (the in-scope namespaces of each element) that does not need to be present explicitly in the ASN.1 representation, so long as there is a way to indicate that this optimized representation is being used in a given instance. The test program effectively assumes that there is a mechanism to signal this, which justifies the general omission of the namespace information items for the in-scope-namespaces.

The test program uses bzip2 compression as an option. Compression is applied only to the single component-strings table and the compressed binary block is included in the abstract value as permitted by the ASN.1 infoset schema. The use of compression results in a significant size reduction at the expense of speed. Two separate sets of speed and size measurements (with and without compression) are provided in Table 4. The ASN.1 specification used in this test is slightly different from the one contained in the current draft of the standard (see Annex A).

3.4 Test results

3.4.1 Discussion of the results

The results shown in Tables 1-4 were obtained by running the test program once on a Dell Latitude C840 notebook computer with a Mobile Intel(R) Pentium(R) 4 - M 1.80 GHz and Windows 2000 Professional Service Pack 3. The use of Aligned vs. Unaligned PER does not seem to affect speed in a significant way. Therefore the speed results in Table 4 can be read as applying equally to Aligned and Unaligned PER. Unaligned PER always produces a more compact encoding than Aligned PER, though the difference is not very large. Table 3 also shows that if the encodings are

compressed with WinZip, the compressed Aligned PER encoding is instead smaller than the compressed Unaligned PER encoding, though by a not very large amount. The use of BER results in encodings that are overly large, usually larger than the XML 1.0 serialization itself. The BER sizes are included in table1 for information. Five XML documents have been used for the test, named as follows:

- much_ado.xml
- periodic.xml
- soap2.xml
- xml.xml
- sfs.xml

The first four XML documents are believed to be the same XML documents that were used for the tests of "XML Stream", published by D. Sosnoski at: http://www.sosnoski.com/opensrc/xmls/. The last XML document is the XML Schema for Schemas, with a portion of the document type declaration removed. Since part of the information in the XML documents is ignored (either because it is not reflected in the infoset or due to the current limitations of the test program), the XML documents used as reference for size comparisons in Tables 1-3 are not the original ones, but those created by the test program itself after a read/write cycle starting from the original documents. It has been verified that further read/write cycles always produce identical XML documents.

Tables 1-3 contain the size results.
The column headings of table1 have the following meanings:

Table 1

Document	XML 1.0	APER		UPER		APER/C		UPER/C	
much_ado.xml	202 044	146 811	*27%*	140 494	*30%*	73 174	*64%*	66 857	*67%*
periodic.xml	113 368	34 630	*69%*	30 471	*73%*	23 217	*80%*	19 058	*83%*
soap2.xml	133 840	47 048	*65%*	39 753	*70%*	40 116	*70%*	32 631	*76%*
xml.xml	157 481	127 467	*19%*	122 286	*22%*	55 414	*65%*	50 231	*68%*
sfs.xml	40 115	17 275	*57%*	15 430	*62%*	11 243	*72%*	9 384	*77%*

- XML 1.0: size of the XML 1.0 serialization (as created by the test program after reading the original XML document);
- APER: size of the Aligned PER encoding of the corresponding ASN.1 infoset schema abstract value;
- UPER: size of the Unaligned PER encoding of the corresponding ASN.1 infoset schema abstract value;
- APER/C: size of the Aligned PER encoding, applying bzip2 compression to the component-strings table;
- UPER/C: size of the Unaligned PER encoding, applying bzip2 compression to the component-strings table;
- BER: size of the BER encoding.

Table 2

Document	XML 1.0	Unique strings	Unique strings compr'd	APER	APER/C	APER minus strings
much_ado.xml	202 044	111 021	37 380	146 811	73 174	35 790
periodic.xml	113 368	15 951	4 533	34 630	23 217	18 679
soap2.xml	133 840	9 419	1 979	47 048	40 116	37 629
xml.xml	157 481	103 567	31 508	127 467	55 414	23 900
sfs.xml	40 115	8 943	2 883	17 275	11 243	8 332

Document	XML 1.0	Unique strings	Unique strings compr'd	UPER	UPER/C	UPER minus strings
much_ado.xml	202 044	111 021	37 380	140 494	66 857	29 473
periodic.xml	113 368	15 951	4 533	30 471	19 058	14 520
soap2.xml	133 840	9 419	1 979	39 753	32 631	30 334
xml.xml	157 481	103 567	31 508	122 286	50 231	18 719

The column headings of Table 2 have the following meanings:

- XML 1.0: same as in Table 1;
- Unique strings: size of the component-strings table;
- Unique strings compr'd: size of the component-strings table, compressed with bzip2;
- APER, APER/C: same as in Table 1;
- APER minus strings: size of the Aligned PER encoding minus the size of the component-strings table (this does not depend on whether bzip2 compression is used);
- UPER, UPER/C: same as in Table 1;
- UPER minus strings: size of the Unaligned PER encoding minus the size of the component-strings table (does not depend on whether bzip2 compression is used).

Table 3

Document	XML 1.0	XML 1.0 /WZ		APER		APER /WZ		APER/C		APER/C /WZ	
much_ado.xml	202 044	53 630	*73%*	146 811	*27%*	57 334	*72%*	73 174	*64%*	47 038	*77%*
periodic.xml	113 368	8 974	*92%*	34 630	*69%*	11 326	*90%*	23 217	*80%*	10 125	*91%*
soap2.xml	133 840	4 896	*96%*	47 048	*65%*	7 514	*94%*	40 116	*70%*	5 805	*96%*
xml.xml	157 481	44 419	*72%*	127 467	*19%*	47 803	*70%*	55 414	*65%*	43 451	*72%*
sfs.xml	40 115	5 654	*86%*	17 275	*57%*	5 698	*86%*	11 243	*72%*	5 837	*85%*

Document	XML 1.0	XML 1.0 /WZ		UPER		UPER /WZ		UPER/C		UPER/C /WZ	
much_ado.xml	202 044	53 630	*73%*	140 494	*30%*	58 939	*71%*	66 857	*67%*	48 289	*76%*
periodic.xml	113 368	8 974	*92%*	30 471	*73%*	13 741	*88%*	19 058	*83%*	12 259	*89%*
soap2.xml	133 840	4 896	*96%*	39 753	*70%*	8 294	*94%*	32 631	*76%*	6 361	*95%*
xml.xml	157 481	44 419	*72%*	122 286	*22%*	53 871	*66%*	50 231	*68%*	46 058	*71%*
sfs.xml	40 115	5 654	*86%*	15 430	*62%*	7 326	*82%*	9 384	*77%*	7 242	*82%*

The column headings of Table 3 have the following meanings:

- XML 1.0: same as in Table 1;
- XML 1.0/WZ: "packed" size as reported by WinZip 8.1 for the XML 1.0 serialization when included in a WinZip archive;
- APER: same as in Table 1;
- APER/WZ: "packed" size as reported by WinZip 8.1 for the Aligned PER encoding when included in a WinZip archive;
- UPER: same as in Table 1;
- UPER/WZ: "packed" size as reported by WinZip 8.1 for the Unaligned PER encoding when included in a WinZip archive;
- APER/C: same as in Table 1;
- APER/C/WZ: "packed" size as reported by WinZip 8.1 for the Aligned PER encoding with a bzip2-compressed component-strings table when included in a WinZip archive;
- UPER/C: same as in Table 1;
- UPER/C/WZ: "packed" size as reported by WinZip 8.1 for the Unaligned PER encoding with a bzip2-compressed component-strings table when included in a WinZip archive.

Table 4 contains the speed results. For each document, the table shows multiple measurements of the duration of the following operations:

Table 4

Document	Operation	Without compression of unique strings						With compression of unique strings				
		M1	M2	M3	M4	Average	Parsing or decoding (indirect)	M1/C	M2/C	M3/C	M4/C	Average/C
much_ado.xml												
	Copy infoset tree	50	49			*50*		51	50			*51*
	Read XML into infoset tree	82	78	83	87	*83*	*33*	83	80	82	84	*82*
	Read PER into infoset tree	71	67	64	66	*67*	*18*	121	116	118	115	*118*
	Write XML from infoset tree	31	29	29	32	*30*		31	29	29	29	*30*
	Write PER from infoset tree	23	24	24	24	*24*		176	177	179	180	*178*
periodic.xml												
	Copy infoset tree	31	35			*33*		31	35			*33*
	Read XML into infoset tree	55	60	61	56	*58*	*25*	56	60	61	59	*59*
	Read PER into infoset tree	40	46	48	44	*45*	*12*	66	72	70	74	*71*
	Write XML from infoset tree	18	19	21	18	*19*		18	17	19	17	*18*
	Write PER from infoset tree	14	15	15	16	*15*		77	78	77	76	*77*
soap2.xml												
	Copy infoset tree	55	52			*54*		55	52			*54*
	Read XML into infoset tree	104	99	101	103	*102*	*48*	104	99	99	106	*102*
	Read PER into infoset tree	72	66	67	68	*68*	*15*	96	91	92	92	*93*
	Write XML from infoset tree	27	27	28	29	*28*		27	27	30	29	*28*
	Write PER from infoset tree	20	20	20	22	*21*		76	74	75	75	*75*
xml.xml												
	Copy infoset tree	37	40			*39*		37	41			*39*
	Read XML into infoset tree	65	69	71	66	*68*	*29*	66	68	71	68	*68*
	Read PER into infoset tree	52	54	57	57	*55*	*17*	99	102	103	100	*101*
	Write XML from infoset tree	22	24	22	22	*23*		22	22	21	22	*22*
	Write PER from infoset tree	18	18	20	18	*19*		167	166	169	166	*167*

- Copy tree: a copy of an infoset tree (previously created from an XML 1.0 document) is created in memory;
- Read XML into infoset tree: an XML 1.0 document is parsed and the corresponding infoset tree is created in memory;
- Read PER into infoset tree: the (Aligned or Unaligned) PER encoding of an ASN.1 infoset schema abstract value is decoded and the corresponding infoset tree is created in memory;
- Write XML from infoset tree: the content of an infoset tree in memory is serialized into an XML 1.0 document;
- Write PER from infoset tree: the content of an infoset tree in memory is converted to an ASN.1 infoset schema abstract value which is then encoded in (Aligned or Unaligned) PER.

The values in Table 4 are durations, expressed in milliseconds, taken from a single execution of the test program. Multiple executions of the test program have resulted in values that are subjectively very close to these values. The column headings of table4 have the following meanings:

- Operation: one of: "Copy tree", "Read XML into infoset tree", "Read PER into infoset tree", "Write XML from infoset tree", "Write PER from infoset tree";
- M1, M2, M3, M4: each of these is an average duration (milliseconds) calculated over a series of 40 invocations of the corresponding operation. Each of the four average values reflects slightly different conditions (the order of invocation of the five operations on the five documents is different in each case). These values are individually reported because they have been observed consistently when running the test program many times.
- Average: the average of M1, M2, M3, and M4.
- Parsing or decoding (indirect): for the "Read XML into infoset tree" operation, the difference between the average time for reading an XML document into an infoset tree and the average time for duplicating an infoset tree in memory. For the "Read PER into infoset tree" operation, the difference between the average time for reading a PER encoding into an infoset tree and the average time for duplicating an infoset tree in memory. These values provide a hint about the relative speed of parsing XML vs. decoding PER, based on the assumption that the time spent to build the infoset tree in memory is approximately the same in both cases.
- M1/C, M2/C, M3/C, M4/C: each of these is an average duration (milliseconds) calculated over a series of 40 invocations of the corresponding operation, with bzip2 compression applied to the component-strings table.
- Average/C: the average of M1/C, M2/C, M3/C, and M4/C.

3.5 Conclusion

The results show a general read and write speed improvement over the processing of XML 1.0 documents when compression is not used.

The read speed improvement is probably limited (in relative terms) by the additional time needed for building the infoset tree in memory, which takes place both when reading an XML document and when reading a PER encoding of the infoset. It would be useful to investigate whether it is possible to implement a SAX interface over the ASN.1 representation avoiding the creation of the infoset tree in memory. In this way, a client application would interact with the ASN.1 infoset processor across a SAX interface and would probably experience a higher speed than when using a regular SAX parser reading an XML 1.0 document.

When the bzip2 compression option is used, the PER read and write speed is lower than the speed of reading and writing XML 1.0, respectively. Since the size reduction due to compression is usually significant, there is a trade-off between speed and the use of compression.

Notice, however, that the ASN.1 infoset schema allows compression to be applied to the unique strings of the infoset, or to a subset of them, or even to multiple subsets independently. Although the test program does not currently support selective use of compression, it would be possible to optimize both size and speed of the ASN.1 representation by applying compression selectively to a subset of the strings. For example, certain larger XML documents may not need to be processed in all of their parts by all reader applications in all cases, so a compressed subset could remain compressed by default and be "lazily" decompressed only when needed. In such cases, there would be a significant size reduction without a corresponding speed penalty.

References

[1] *XML Information Set*, W3C Recommendation - 24 October 2001, http://www.w3.org/TR/xml-infoset.

[2] *Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*, International standard ITU-T Rec. X.680 (2002) | ISO/IEC 8824-1:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf.

[3] *Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification*, International standard ITU-T Rec. X.681 (2002) | ISO/IEC 8824-2:2002,

http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf.

[4] *Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification*, International standard ITU-T Rec. X.682 (2002) | ISO/IEC 8824-3:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.682-0207.pdf.

[5] *Information Technology - Abstract Syntax Notation One (ASN.1) : Parameterization of ASN.1 Specifications*, International standard ITU-T Rec. X.683 (2002) | ISO/IEC 8824-4:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.683-0207.pdf.

[6] *Information Technology - Abstract Syntax Notation One (ASN.1) : Encoding Rules : Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*, International standard ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf.

[7] *Information Technology - Abstract Syntax Notation One (ASN.1): ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)*, International standard ITU-T Rec. X.691 (2002) | ISO/IEC 8825-2:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.691-0207.pdf.

[8] *Information Technology - Abstract Syntax Notation One (ASN.1): ASN.1 Encoding Rules: Specification of Encoding Control Notation (ECN)*, International standard ITU-T Rec. X.691 (2002) | ISO/IEC 8825-2:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.692-0207.pdf.

[9] *Information Technology - Abstract Syntax Notation One (ASN.1): ASN.1 encoding rules: XML encoding rules (XER)*, International standard ITU-T Rec. X.693 (2001) | ISO/IEC 8825-4:2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.693-0112.pdf.

Appendix A ASN.1 specification used in the tests of the ASN.1 Schema for the XML Information Set

The ASN.1 specification used in this test is slightly different from the one contained in OSS Nokalva's contribution to the ISO/ITU-T. The reason of this is that the creation of the test program began before the contribution was submitted and because a few small changes have been made to the ASN.1 specification but are yet to be reflected in the ISO/ITU-T document. However, it is believed that the differences in the ASN.1 specifications, when the ISO/ITU-T document is updated, will not result in significant differences in the speed and size of the PER encoding.

```
Infoset DEFINITIONS AUTOMATIC TAGS ::= BEGIN

--Referenced by: none--
Document ::= SEQUENCE {
  children SEQUENCE OF CHOICE {
    element Element,
    processing-instruction ProcessingInstruction,
    comment Comment,
    document-type-declaration DocumentTypeDeclaration },

  notations SEQUENCE OF Notation,
  unparsed-entities SEQUENCE OF UnparsedEntity,
  base-URI URIOrIndex OPTIONAL,
  character-encoding-scheme StringOrIndex,
  standalone BOOLEAN OPTIONAL,
  version StringOrIndex OPTIONAL,
  all-declarations-processed BOOLEAN,

  component-strings SEQUENCE OF SEQUENCE {
    compression-algorithm URI,
    component-strings-compressed-block OCTET STRING },

  common-names SEQUENCE OF SEQUENCE {
    namespace-name CHOICE {
      namespace-name URIOrIndex,
      same-as-previous-item NULL } OPTIONAL,

    local-name NCNameOrIndex,

    prefix CHOICE {
      prefix NCNameOrIndex,
      same-as-previous-item NULL } OPTIONAL }
}

--Referenced by: Document, Element--
Element ::= SEQUENCE {
  name CHOICE {
    name SEQUENCE {
      namespace-name CHOICE {
```

```
        namespace-name URIOrIndex,
        same-as-parent-element NULL } OPTIONAL,

      local-name NCNameOrIndex,
      prefix NCNameOrIndex OPTIONAL },

    common-name-index SmallIndex },

  children CHOICE {
    element-content-characters StringOrIndex,

    children SEQUENCE (SIZE(0 .. 3, ..., 0 .. MAX)) OF CHOICE {
      element Element,
      processing-instruction ProcessingInstruction,
      unexpanded-entity-reference UnexpandedEntityReference,
      characters Characters,
      comment Comment } },

  attributes SEQUENCE (SIZE(1 .. 4, ..., 1 .. MAX)) OF Attribute
OPTIONAL,
  namespace-attributes SEQUENCE (SIZE(1 .. 4, ..., 1 .. MAX)) OF
Attribute OPTIONAL,

  in-scope-namespaces CHOICE {
    in-scope-namespaces SEQUENCE OF Namespace,
    same-as-parent-element NULL },

  base-URI CHOICE {
    base-URI URIOrIndex,
    same-as-parent-element NULL } OPTIONAL
}

--Referenced by: Element--
Attribute ::= SEQUENCE {
  name CHOICE {
    name SEQUENCE {
      namespace-name CHOICE {
        namespace-name URIOrIndex,
        same-as-owner-element NULL,
        same-as-previous-attribute NULL } OPTIONAL,

      local-name NCNameOrIndex,
      prefix NCNameOrIndex OPTIONAL },

    common-name-index SmallIndex },

  normalized-value StringOrIndex,
  specified BOOLEAN OPTIONAL,

  attribute-type CHOICE {
    attribute-type AttributeType,
    unknown NULL } OPTIONAL,

  references CHOICE {
    element-indexes SEQUENCE OF RELATIVE-OID,
    unparsed-entity-indexes SEQUENCE OF SmallIndex,
    notation-index SmallIndex,
```

```
    unknown NULL } OPTIONAL
}

--Referenced by: Document, Element, DocumentTypeDeclaration--
ProcessingInstruction ::= SEQUENCE {
  target NCNameOrIndex,
  content StringOrIndex,

  base-URI CHOICE {
    base-URI URIOrIndex,
    same-as-parent NULL },

  notation CHOICE {
    notation-index SmallIndex,
    unknown NULL } OPTIONAL
}

--Referenced by: Element--
UnexpandedEntityReference ::= SEQUENCE {
  name NCNameOrIndex,

  system-identifier CHOICE {
    system-identifier URIOrIndex,
    unknown NULL } OPTIONAL,

  public-identifier CHOICE {
    public-identifier URIOrIndex,
    unknown NULL } OPTIONAL,

  declaration-base-URI CHOICE {
    declaration-base-URI URIOrIndex,
    unknown NULL } OPTIONAL
}

--Referenced by: Element--
Characters ::= SEQUENCE {
  characters CHOICE {
    characters StringOrIndex,
    binary-block OCTET STRING },

  element-content-whitespace BOOLEAN
}

--Referenced by: Document, Element--
Comment ::= SEQUENCE {
  content StringOrIndex
}

--Referenced by: Document--
DocumentTypeDeclaration ::= SEQUENCE {
  system-identifier URIOrIndex OPTIONAL,
  public-identifier URIOrIndex OPTIONAL,
  children SEQUENCE OF ProcessingInstruction
}

--Referenced by: Document--
UnparsedEntity ::= SEQUENCE {
```

```
  name NCNameOrIndex,
  system-identifier URIOrIndex OPTIONAL,
  public-identifier URIOrIndex OPTIONAL,
  declaration-base-URI URIOrIndex OPTIONAL,
  notation-name NCNameOrIndex,

  notation CHOICE {
    notation-index SmallIndex,
    unknown NULL } OPTIONAL
}

--Referenced by: Document--
Notation ::= SEQUENCE {
  name NCNameOrIndex,
  system-identifier URIOrIndex OPTIONAL,
  public-identifier URIOrIndex OPTIONAL,
  declaration-base-URI URIOrIndex OPTIONAL
}

--Referenced by: Element--
Namespace ::= SEQUENCE {
  prefix NCNameOrIndex OPTIONAL,
  namespace-name URIOrIndex
}

StringOrIndex ::= CHOICE {
  string UTF8String,
  component-string-index LargeIndex }

URIOrIndex ::= CHOICE {
  uri URI,
  component-string-index LargeIndex }

NCNameOrIndex ::= CHOICE {
  ncname NCName,
  component-string-index LargeIndex }

NCName ::= UTF8String

URI ::= UTF8String

SmallIndex ::= INTEGER (0 .. 15, ..., 0 .. MAX)

LargeIndex ::= INTEGER (0 .. 2047, ..., 0 .. MAX)

AttributeType ::= ENUMERATED {
  id,
  idref,
  idrefs,
  entity,
  entities,
  nmtoken,
  nmtokens,
  notation,
  cdata,
  enumeration
}
```

END