Measuring the Capacity of an XML Schema

Specifying an Information Channel with an XML Schema

August 2006

Roger L. Costello[1], The MITRE Corporation
Robin A. Simmons[2], The MITRE Corporation

[1] Roger L. Costello, The MITRE Corporation, 202 Burlington Rd., Bedford, MA 01730
[2] Robin A. Simmons, The MITRE Corporation, 7515 Colshire Drive, McLean, VA 22102

Abstract

Fundamental concepts for measuring the amount of information specified by an Extensible Markup Language (XML) Schema are presented. A web service or XML application can use an XML Schema to specify the form of XML data allowed to invoke the service or application. The XML Schema may be viewed as a specification of an information channel to the web service or application. The capacity of the information channel specified by the XML Schema can be numerically measured.

Knowledge of the capacity of a channel specified by an XML Schema is beneficial for web services and XML applications. The capacity indicates how much effort might be required by a web service or application to process XML instances, thus serving as an indicator of the potential load on the web service or application.

Knowledge of the capacity of a channel specified by an XML Schema is also beneficial for security services that regulate against undesirable information exchanges. The amount of regulation required is proportional to the variety of information received. Thus, XML data conforming to an XML Schema that has a large capacity will require a comparatively large effort to regulate. As a general rule, the higher the capacity, the larger the security risk.

1 Introduction

1.1 Purpose

Information Theory[3] concepts are applied to Extensible Markup Language (XML) Schemas[4], treating an XML Schema as a specification of an information channel. The goal is to measure the maximum capacity of any XML instance document that conforms to the XML Schema. The result is a number that represents the capacity, in bits, of the XML Schema.

Suppose a web service is invoked by sending it XML instance documents that conform to an XML Schema. The XML Schema may be viewed as defining an information channel to the web service. The XML Schema specifies the form and size of the channel. The web service is invoked by sending information through the channel.

A channel has a certain capacity, that is, a certain quantity of information that it can carry. A channel can carry less than its capacity, but it cannot carry more than its capacity.

Applying this Information Theory concept to an XML Schema provides an approach for determining the maximum capacity of a channel that is specified by an XML Schema.

1.2 Background

A blank sheet of paper can hold a number of characters of a specified font and size. The amount of information that the sheet of paper can contain is limited to the capacity of the paper. Increasing the size of the paper or the number of sheets of paper increases the amount of information that can be carried. Similarly, reducing the size of the paper, the number of characters, or the allowable character set will reduce the capacity of the information that can be carried.

The capacity of a sheet of paper can be measured. The quantitative measurement of this capacity provides a value of how much reading, writing, or storage might be required to process the information that can be contained on the paper.

It is useful to quantitatively measure the amount of information that could be transmitted over a channel or written on a sheet of paper. This number provides a concrete value of:

- The amount of storage that might be required.
- The amount of time to read or write the information.
- The expected complexity of input validation.
- The amount of time to process an input.
- The opportunity for the information to contain malicious content.

[3] Shannon, Weaver, 1949.
[4] Thompson et al, 2004.

Similarly, it is useful to measure the capacity of an XML Schema. This provides a quantitative measure of the amount of information that might be included in an XML instance document that conforms to an XML Schema.

1.3 Document Organization

This document has five sections. This section provides an introduction and background. Section 2 presents two simple examples to demonstrate how the capacity of XML Schemas can be measured. Section 3 discusses the meaning of the capacity measurement, and what might occur if the capacity is "too large." Section 4 introduces terminology and assumptions, recommends how to measure the capacity of the XML Schema components, including the XML built-in datatypes, and explains how to combine the component measurements into a meaningful measurement for an entire XML Schema. Section 5 provides a summary.

2 Measuring the Information Capacity of XML Schemas

An XML Schema describes the format and content of XML instance documents. The information capacity of an XML Schema represents the number of different data values that an XML instance document can contain if the instance document strictly conforms to the XML Schema. This section provides two examples that demonstrate measuring the information capacity of a channel specified by an XML Schema.

2.1 First Example: A Pair of Integers

Consider the following excerpt from an XML Schema. It defines a simple datatype called "Four-Integers" whose value space are the integers 1, 2, 3, or 4, and it defines a root element called <numbers> that contains two child elements <a> and <b> that can each contain data specified by the "Four-Integers" datatype.

(Example 1)

```
<simpleType name="Four-Integers">
    <restriction base="integer">
        <minInclusive value="1"/>
        <maxInclusive value="4"/>
    </restriction>
</simpleType>
<element name="numbers">
    <complexType>
        <sequence>
            <element name="a" type="Four-Integers"/>
            <element name="b" type="Four-Integers"/>
        </sequence>
    </complexType>
</element>
```

The following sample XML instance document conforms to the above XML Schema:

```
<numbers>
    <a>2</a>
    <b>4</b>
</numbers>
```

An XML instance document that conforms to this XML Schema can contain any of 16 pairs {<a>,<b>} of integers: {1,1}, {1,2}, {1,3}, {1, 4}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {3,1}, {3,2}, {3,3}, {3, 4}, {4,1}, {4,2}, {4,3}, and {4, 4}. Exactly 16 canonical[5] XML instance documents will conform to this XML Schema. All other XML instance documents will not conform to the

[5] XML Canonicalization is a process of normalizing the representation of XML documents to determine whether two XML documents are the same. See Boyer et al, 2001.

XML Schema and can be considered invalid. The channel specified by this XML Schema must be capable of carrying any of the 16 pairs of values.

2.2 Second Example: A Text String

Consider the following excerpt from another XML Schema. It defines a root element called "text" that has one child element "m" that contains any text string.

```
<element name="text">                                    (Example 2)
    <complexType>
        <sequence>
            <element name="m" type="string"/>
        </sequence>
    </complexType>
</element>
```

An XML instance document that conforms to this XML Schema will have an element <m> that can contain a limitless number of characters in a string. The channel that is specified by this XML Schema must be capable of carrying a string of any length with any characters. In other words, the capacity of this XML Schema is infinite.

2.3 Expressing the Capacity Logarithmically

The value of the capacity measurement is often large, so typically it is more convenient to express channel capacity on a base 2 logarithmic scale. When expressed this way, the capacity is given the units *bits*. In Example 1, the channel capacity is $\log_2(16) = 4$ bits. In Example 2, the channel capacity is infinite. In this paper, capacity will be expressed logarithmically, in bits.

2.4 Implications of XML Schema Capacity

It is important to note that measuring the capacity of an information channel is a means to measure the *variety of data values*. It is not a measure of how a specific computer physically represents the data. In Example 1, the variety of data values that can be contained in an XML instance document is 16, expressed as 4 bits.

A computer physically represents each integer during storage or processing using some number of bytes. The XML Schema capacity is not a measurement of the number of bytes used by a computer to store or process the integer. Furthermore, data in XML format has additional overhead due to the tags. For example, it is unlikely (although possible) that a specific computer can use 4 bits to represent the allowable {<a>,<b>} values described in Example 1. The capacity measurement represents the variety of allowable data values but provides no information about how the channel is physically implemented.

To summarize, by treating an XML Schema as a specification of an information channel, one can measure the capacity of the XML Schema. This measurement is equivalent to the capacity of XML instance documents that strictly conform to the XML Schema.

3 Using the Capacity Measurement

This section discusses the usefulness of the capacity measurement and provides a basic framework for using the capacity measurement to make comparisons. A quantitative measurement by itself provides some value, but comparing quantitative measurements, either against a threshold or against a measurement associated with competing designs or products, can provide additional value. The capacity measurement helps create an objective comparison of the benefit versus cost of competing designs or products.

3.1 How to Use a Capacity Measurement

A quantitative measurement of the capacity of an XML Schema would be valuable in:

- Assessing the load on web services or XML applications.
- Comparing the complexity associated with competing designs.
- Measuring the expected efficacy of an XML-based security capability.
- Determining the suitability of using an XML Schema to enforce format or content constraints.
- Determining the risk of inserting malicious or unauthorized data into a web service or an XML application.

These uses assume that the XML Schema is used to constrain the data received over a data channel by the web service or XML application.

For example, consider the XML Schema excerpt presented in Example 1. If the restrictions that limit the minimum and maximum inclusive values are expanded to 0 and 9, respectively, then the two elements <a> and <b> can contain any of the ten digits (zero through nine). This increases the information capacity from 4 bits (16 unique pairs of values) to 6.64 bits (100 unique pairs of values), which represents a 525 percent increase.

3.2 How Much Capacity Is Too Much?

An important security principle is that information systems must validate data received from untrusted sources.[6] Ashby's Law of Requisite Variety[7] says that regulation of an input requires an amount of variety at least equal to the variety of the input. Although Ashby is referring to a control system, the implication for data validation is that the complexity of the validation process increases as the complexity of the data increases. If the capacity of an XML Schema is infinite (as in Example 2 above), it might be impractical to validate the data automatically. Such an XML Schema might be inappropriate for use in an unprotected web service or in a security system.

Policy or operational constraints associated with a web service or XML-based security solution might require that the channel capacity be constrained to a finite value. The capacity

[6] Bishop, 2005.
[7] Ashby, 1961.

measurement can help determine if an XML Schema is capable of supporting the policy or operational constraints.

4 Measuring the Capacity of an XML Schema

This section provides assumptions and defines terminology that can be helpful when measuring the capacity of an XML Schema. It then shows how to calculate the capacity of the built-in XML datatypes, unions of datatypes, lists, recurring elements, complex types, namespaces, and other components of an XML Schema. It concludes by showing how to combine the capacity measurements of the individual XML Schema components into an aggregate capacity measurement for the entire XML Schema.

4.1 Assumptions

These assumptions will be made when measuring the capacity of an XML Schema:

- The XML Schema defines at least one global element that can be used as the root element of XML instance documents.
- If the XML Schema defines more than one global element, meaning there are several potential root elements for an XML instance document, then the capacity measurement for the XML Schema will result in a capacity number for each global element.
- If the XML Schema references a namespace, then the XML Schema or Schemas defining that namespace must be accessible.
- The set of allowable characters consists of the 7-bit American Standard Code for Information Interchange (ASCII) characters. Thus, there are 128 possible characters. Although XML allows use of the full Unicode set of characters, this paper discusses only the ASCII character set to limit the capacity measurement problem.

4.2 Terminology

Throughout this document, the phrase “capacity of an XML Schema” will sometimes be used as shorthand to indicate the capacity of the channel specified by the XML Schema. Similarly, the phrase “capacity of an XML element” refers to the capacity of the channel that is specified by the element declaration in the XML Schema.

Section 4.3 recommends how to determine the capacity of the XML Schema datatypes and complex types. Technically, datatypes do not have a capacity; only elements and attributes have capacity. However, the capacity of an element or attribute is determined by its datatype. Consequently, it will be useful to measure (and refer to) the capacity of datatypes and complex types.

Similarly, Section 4.3.10 discusses the capacity of a namespace. A namespace does not contain values, so namespaces do not have a capacity. However, the XML Schema or Schemas associated with the namespace are likely to define elements and datatypes. To calculate the capacity of an XML Schema that incorporates an element or datatype from another namespace *N*, one must first compute the maximum capacity of the elements in namespace *N*. This is stated informally as computing the capacity of namespace *N*.

4.3 Calculating the Capacity of an XML Schema

The following subsections describe how to calculate the capacity of an XML Schema. The first subsections show how to measure the capacity of the lowest level XML Schema components. Subsequent subsections show how to combine the capacity measurements of the lowest level components, culminating in an approach for measuring the capacity of the XML Schema as a whole.

4.3.1 Capacity of the XML Schema Built-in Datatypes

The capacity of a datatype *D* is equal to the base 2 logarithm of the cardinality of the value space of *D*. That is, a datatype can contain a certain number of values. The capacity of the datatype is the (base 2) logarithm of that number of values.

Note that a datatype can have an infinite value space (for example, an integer) but a computer may not support all possible values. So in practice, the value space is constrained by the limits of the computer. However, the capacity measurement is not concerned with specific implementations or machine limitations and therefore can result in an infinite capacity measurement.

Table 1 describes the value space of each of the built-in XML datatypes[8] and recommends how to calculate the cardinality and capacity of each datatype.

[8] Biron et al, 2004.

Table 1. Capacity of the Built-In XML Datatypes

Built-In Datatype	Value Space	Cardinality	Capacity (in bits)
anySimpleType	an unbounded sequence of characters	128^s where s = number of characters	7s
anyURI	[a-zA-Z0-9/#.-@;=+$,_~*!'%?:&]*	80^s where s = number of characters	6.32s
base64Binary	[A-Za-z0-9+/]*	64^s where s = number of characters	6s
boolean	0, 1, true, false	4	2
byte	–127 to 128	256	8
date	(+/–)yyyy-mm-dd+/-hh:00	2 × 9999 × 365 × 14 = 102,189,780	26.6
dateTime	(+/–)yyyy-mm-ddThh:mm:ss.sss+/-hh:00	2 × 9999 × 365 × 24 × 60 × 60 × 1000 × 14 = 8,829,196,992,000,000	52.97
decimal	(+/–)dddddd.sssss	10^{d+s} where d = number of digits to the left decimal point and s = number of digits to the right decimal point	3.32(d + s)
double	(+/–)m × 2^e where m is 0 to 2^{53} and e is –1075 to 970	2 × 2^{53} × 2045	1 + 53 + 11 = 65
duration	P*n*Y*n*M*n*DT*n*H*n*M*n*S.zzzzzz	$10^y \times 10^m \times 10^d \times 24 \times 60 \times 60 \times 10^z =$ $86400 \times 10^{y+m+d+z}$ where y = number of year digits, m = number of month digits, d = number of day digits, and z = number of digits to the right of the decimal point	16.4 + 3.32(y + m + d + z)
ENTITY	[a-zA-Z_][a-zA-Z0-9.-_]*	65^s where s = number of characters	6.02s
ENTITIES	[a-zA-Z_][a-zA-Z0-9.-_]*	66^s Let s = number of characters	6.04s

Built-In Datatype	Value Space	Cardinality	Capacity (in bits)
float	$m \times 2^e$ where m is -2^{24} to 2^{24} and e is –149 to 104	$2 \times 2^{24} \times 253 = 2^{33}$	33
hexBinary	[0-9a-fA-F]*	22^d where d = number of hex digits	4.46d
gDay	--01 to --31	31	4.95
gMonth	--01 to --12	12	3.58
gMonthDay	--01-01 to --12-31	365	8.51
gYear	(+/–)0001 to 9999	2 × 9999 = 19998	14.29
gYearMonth	(+/–)0001-01 to 9999-12	2 × 12 × 9999 = 19998	17.87
ID	[a-zA-Z0-9.-_]*	65^s where s = number of characters	6.02s
IDREF	[a-zA-Z_][a-zA-Z0-9.-_]*	65^s where s = number of characters	6.02s
IDREFS	[a-zA-Z_][a-zA-Z0-9.-_]*	66^s where s = number of characters	6.04s
int	–2147483648 to 2147483647	4294967296	32
integer	(+/–)ddddddd	2×10^d where d = number of digits	1 + 3.32d
language	unbounded sequence of characters	128^s where s = number of characters	7s
long	–9223372036854775808 to 9223372036854775807	18,446,744,073,709,551,614	64
Name	[a-zA-Z_:][a-zA-Z0-9.-_:]*	66^s where s = number of characters	6.04s
NCName	[a-zA-Z_][a-zA-Z0-9.-_]*	65^s where s = number of characters	6.02s
negativeInteger	–ddddddd	10^d where d = number of digits	3.32d
NMTOKEN	[a-zA-Z0-9.-_:]*	66^s where s = number of characters	6.04s
NMTOKENS	[a-zA-Z0-9.-_:]*	67^s where s = number of characters	6.07s
nonNegativeInteger	ddddddd	10^d where d = number of digits	3.32d
nonPositiveInteger	–ddddddd	10^d where d = number of digits	3.32d

Built-In Datatype	**Value Space**	**Cardinality**	**Capacity (in bits)**
normalizedString	unbounded sequence of characters, except carriage return, line feed, and tab	125^s where s = number of characters	6.97s
NOTATION	[a-zA-Z_][a-zA-Z0-9.-_]*	65^s where s = number of characters	6.02s
positiveInteger	ddddddd	10^d where d = number of digits	3.32d
QName	[a-zA-Z_][a-zA-Z0-9.-_]*	65^s where s = number of characters	6.02s
short	–32768 to 32767	65536	16
string	unbounded sequence of characters	128^s where s = number of characters	7s
time	hh:mm:ss.sss-hh	24 × 60 × 60 × 1000 × 14 = 1,209,600,000	30.17
token	unbounded sequence of characters, except line-feed, carriage return, and tab and cannot start or end with a space, and cannot have consecutive spaces (approximated by assuming no spaces)	124^s where s = number of characters	6.95s
unsignedByte	0 to 255	256	8
unsignedInt	0 to 4294967295	4,294,967,296	32
unsignedInteger	0 to ddddddddd	10^d where d = number of digits	3.32d
unsigned Long	0 to 18446744073709551615	18446744073709551616	64
unsigned Short	0 to 65535	65536	16

As can be seen from Table 1, the capacity of some of the built-in datatypes can become quite large. In some cases, the capacity is a function of the number of characters or digits in the datatype. From Ashby's Law of Requisite Variety we know that a large capacity requires a correspondingly large variety to regulate it.

The creator of an XML Schema can define simple datatypes that restrict the number of characters or digits in some of the datatypes. This definition allows the XML Schema to constrain the value space of some datatypes, which might be desirable for certain web services or XML applications. The next section explains how to determine the capacity of simple datatypes.

4.3.2 Capacity of a User-Defined Simple Datatype

The XML Schema Specification allows for user-defined datatypes, called a simpleType. A user-defined datatype is derived from an existing datatype by constraining properties (called *facets*) of the existing datatype. The first subsection provides examples of user-defined simple datatypes and explains how to determine their capacity. The subsequent subsection provides an approach for determining the capacity of any user-defined simple datatype.

4.3.2.1 Examples of User-Defined Simple Datatypes

Users can use XML Schema facets, such as "minInclusive" and "maxInclusive", to restrict the value space of a built-in XML datatype. Restricting a datatype decreases the capacity of the information channel. The following three examples demonstrate how to reduce the capacity of a built-in datatype.

(Example 3)

```
<simpleType name="Example3">
    <restriction base="integer">
        <minInclusive value="-20"/>
        <maxInclusive value="20"/>
    </restriction>
</simpleType>
```

The value space for this user-defined simpleType is the integers from –20 to +20. It permits 41 possible values. Logarithmically, the capacity is $\log_2(41) = 5.36$ bits. This user-defined simple datatype has reduced the capacity of the integer datatype from 1 + 3.32d bits (per Table 1), which equates to 7.64 bits for a two-digit integer, to 5.36 bits.

The next example creates a user-defined simpleType that enumerates four specific values for the byte datatype.

(Example 4)

```
<simpleType name="Example4">
    <restriction base="byte">
```

```
            <enumeration value="12"/>
            <enumeration value="5"/>
            <enumeration value="88"/>
            <enumeration value="47"/>
        </restriction>
    </simpleType>
```

The user-defined simpleType in Example 4 permits 4 values: 12, 5, 88, or 47. In other words, the cardinality of this simpleType is 4. Logarithmically, its capacity is 2 bits.

The final example restricts the value space by specifying the number of digits in a byte dataType.

```
    <simpleType name="Example5">                                    (Example 5)
        <restriction base="byte">
            <totalDigits value="2"/>
        </restriction>
    </simpleType>
```

This user-defined simpleType permits any signed or unsigned 2-digit byte value from –99 to +99. The cardinality of this simpleType is $2 \times 10^2 = 200$. As stated in Example 3, the capacity of a two-digit integer is 7.64 bits.

4.3.2.2 General Algorithm for Computing the Cardinality of a User-Defined Numeric Datatype

It is possible to develop a general algorithm for computing the capacity of any simpleType that uses a numeric datatype. A numeric datatype[9] can be constrained via these facets (Table 2):

[9] Biron et al, 2004.

Table 2. Description of the Numeric Facets

Facet Name	Purpose
totalDigits	Restricts the total number of digits; the digits to the right of the decimal point are included in this facet value.
fractionDigits	Restricts the number of digits to the right of the decimal point.
pattern	Restricts the value through the use of a regular expression.
whiteSpace	This facet has no effect on numeric values.
enumeration	Restricts the value by enumerating the allowed values.
maxInclusive	Restricts the maximum value, where the value specified is included.
maxExclusive	Restricts the maximum value, where the value specified is excluded.
minInclusive	Restricts the minimum value, where the value specified is included.
minExclusive	Restricts the minimum value, where the value specified is excluded.

The following algorithm is proposed for computing the cardinality of a simpleType that restricts a base datatype that is numeric:

1. Compute the value range from the facets that are present:

```
Initialize lo and hi to the minimum and maximum value for the base datatype.  For
example, if the base datatype is byte, then initialize lo = –128, hi = 127;

If minInclusive facet is present, then lo = max(lo, minInclusive);
If maxInclusive facet is present, then hi = min(hi, maxInclusive);
If minExclusive facet is present, then lo = max(lo, minExclusive + 1);
If maxExclusive facet is present, then hi = min(hi, maxExclusive + 1);

If totalDigits facet is present, then
    If fractionDigits facet is present, then
        $lo = \max(lo, -10^{\text{totalDigits - fractionDigits}} + 1)$,
        $hi = \min(hi, 10^{\text{totalDigits - fractionDigits}} - 1)$;
    else
        $lo = \max(lo, -10^{\text{totalDigits}} + 1)$,
        $hi = \min(hi, 10^{\text{totalDigits}} - 1)$;

Else (totalDigits facet is not present)
    If fractionDigits facet is present, then
        If number-of-fractional-digits-in(lo) < fractionDigits then lo = lo;
        Else lo = truncate lo to have fractionDigits to the right of the decimal point;
            If number-of-fractional-digits-in(hi) < fractionDigits, then
                hi = add to hi a decimal value so that it has fractionDigits to the right
                    of the decimal point;
```

Else *hi* = truncate *hi* to have fractionDigits to the right of the decimal point.

2. If enumeration facets are present:

   Cardinality is the number of enumeration values within the range [*lo*…*hi*] that also satisfy any pattern values.

3. If regular expression patterns are present:

   Cardinality is the number of values within the range [*lo*…*hi*] that also satisfy the patterns.

4. If *lo* and *hi* are whole numbers then cardinality is $hi - lo + 1$;
   Otherwise,
   lsd = maximum digits to the right of the decimal point (*lo*, *hi*)
   cardinality = $hi \times 10^{lsd} - lo \times 10^{lsd} + 1$.

Subsection 4.3.3 generalizes this algorithm to determine the capacity of a restriction of any built-in datatype or user-defined simpleType.

4.3.2.3 General Algorithm for Computing the Cardinality of a User-Defined Text Datatype

It is possible to develop a general algorithm for computing the cardinality of any simpleType that uses a text datatype. A text datatype[10] can be constrained via the facets listed in Table 3.

Table 3. Description of the Text Facets

Facet Name	Purpose
length	Restricts the total number of characters.
minLength	Restricts the minimum number of characters.
maxLength	Restricts the maximum number of characters.
pattern	Restricts the value through the use of a regular expression.
enumeration	Restricts the value by enumerating the allowed values.
whitespace	Restricts the amount of whitespace (space, tab, linefeed, carriage return) in the value.

The following algorithm is recommended for computing the cardinality of a simpleType that restricts a base datatype that is text:

[10] Biron et al, 2004.

1. Compute the range of the length of the text from the facets that are present:

   Initialize *lo* and *hi* to the minimum and maximum length for the base datatype. For example, if the base datatype is string then initialize *lo* = 0, *hi* = infinity;

   If length facet is present, then *lo* = length, *hi* = length;

   If minLength facet is present, then *lo* = minLength;

   If maxLength facet is present, then *hi* = maxLength;

2. If enumeration facets are present:

   Cardinality is the number of enumeration values within the range [*lo*…*hi*]
   that also satisfy any pattern values;

3. If regular expression patterns are present:

   Cardinality is the number of values within the range [*lo*...*hi*]
   that also satisfy the patterns;

4. Otherwise, cardinality is *hi* – *lo* + 1.

The whitespace facet is not treated. Its effect is specific to the XML instance document, and no meaningful measure can be made from the XML Schema.

4.3.3 Capacity of a Numeric User-Defined Simple Datatype That Restricts another SimpleType

This is a more general algorithm for determining the capacity of a restriction of numeric datatype. The simpleType being restricted can be any built-in numeric data type or user-defined simpleType. This recursive algorithm is presented in a pseudo-code format.

```
Function ComputeCapacity (simpleType)

    If simpleType base type is a primitive type, then
        Compute the cardinality using the algorithm in Section 4.3.2.2.

    Else
        new simpleType = the base type identified in simpleType;
        {lo, hi} = ComputeRange (new simpleType);
        Compute the cardinality using the algorithm in Section 4.3.2.2
            using {lo, hi} as the initial range.
    Return cardinality;


Function ComputeRange (simpleType)

    If simpleType base type is a primitive type, then
        lo = the minimum value for the primitive type;
        hi = the maximum value for the primitive type;
        Return ComputeRestrictedRange (simpleType, lo, hi);

    Else
        new simpleType = the base type identified in simpleType;
        {lo, hi} = ComputeRange (new simpleType);
    Return ComputeRestrictedRange (simpleType, lo, hi).


Function ComputeRestrictedRange (simpleType, lo, hi)

    If minInclusive facet is present, then lo = max(lo, minInclusive);
    If maxInclusive facet is present, then hi = min(hi, maxInclusive);
    If minExclusive facet is present, then lo = max(lo, minExclusive + 1);
    If maxExclusive facet is present, then hi = min(hi, maxExclusive + 1);

    If totalDigits facet is present, then
        If fractionDigits facet is present, then
            $lo = \max(lo, -10^{\text{totalDigits - fractionDigits}} + 1)$,
            $hi = \min(hi, 10^{\text{totalDigits - fractionDigits}} - 1)$;
        Else
            $lo = \max(lo, -10^{\text{totalDigits}} + 1)$,
            $hi = \min(hi, 10^{\text{totalDigits}} - 1)$;
    Else (totalDigits facet is not present)
        If fractionDigits facet is present, then
            If number-of-fractional-digits-in(lo) < fractionDigits, then lo = lo
            Else lo = truncate lo to have fractionDigits to the right of the decimal point;
            If number-of-fractional-digits-in(hi) < fractionDigits, then
                hi = add to hi a decimal value so that it has fractionDigits to the right of
                of the decimal point;
            Else hi = truncate hi to have fractionDigits to the right of the decimal point;

    Return {lo, hi}.
```

Convert the cardinality value to the capacity by calculating the base two logarithm of the cardinality value.

4.3.4 Capacity of an Element with a Datatype That Is a Simple Type

The capacity of an element is the capacity of the simple type:

<element name="*E*" type="*D*"/>

capacity(*E*) = capacity(*D*)

4.3.5 Capacity of an Element with Multiple Occurrences

An element may be declared to allow multiple occurrences:

<element name="*E*" type="*D*" maxOccurs="*i*"/>

The capacity of *E* is the capacity of a single occurrence raised to the *i* power. In logarithmic scale, this equates to multiplying *i* by the logarithmic capacity of a single occurrence of E:

capacity(all occurrences of E) = $i \times$ capacity(E)

Here's an example:

<element name="*E*" type="boolean" maxOccurs="2"/>

The capacity of a single occurrence of *E* is 2 bits. The capacity of two occurrences of *E* is 2×2 bits = 4 bits.

4.3.6 Capacity of a Union Datatype

The capacity of a union type is the sum of the logarithmic capacity of each item type:

<union memberTypes="D1 D2 D3"/>

capacity(union type) = capacity(D1) + capacity(D2) + capacity(D3)

This approach applies whether the union type is defined via a <union> or via nested simpleTypes. The capacity of nested simpleTypes is the sum of the logarithmic capacity of each simpleType.

4.3.7 Capacity of a List Datatype

The capacity of a list type is the logarithmic capacity of each item type multiplied by the number of allowable items:

Let s = the number of items in the list.

Let d = the datatype of each item.

capacity(list type) = s × capacity(d)

4.3.8 Capacity of Attributes

Attributes can have only a simple datatype value. So, the capacity of an attribute is the capacity of its datatype.

<attribute name="*A*" type="*D*"/>

capacity(*A*) = capacity(*D*)

4.3.9 Capacity of a Complex Type

A complex type is a user-defined datatype that contains other elements, attributes, or complex types, or virtually any combination of elements, attributes, and complex types. The following subsections recommend how to determine the capacity of a number of cases of complex types.

4.3.9.1 Capacity of a Complex Type That Contains Two or More Child Elements

The value space created by a complex type created from a sequence of child elements is determined by multiplying the value space of each of the child elements. Logarithmically, the capacity of the complex type is calculated by adding the capacity of each child element.

```
<complexType name="C">
    <sequence>
        <element name="E1" type="D1"/>
        <element name="E2" type="D2"/>
        . . .
    </sequence>
</complexType>
```

The capacity of a <sequence> complexType is the sum of the logarithmic capacity of each child element:

capacity(*C*) = capacity(*E1*) + capacity(*E2*) + …

Note: Using <all> instead of <sequence> yields the same capacity because order is irrelevant when adding the capacity of child elements together.

4.3.9.2 Capacity of a Complex Type with a Choice of Two or More Child Elements

The value space created by a complex type created from a choice of one child element from a list of child elements is simply the value space of the child element with the largest value space.

```
<complexType name="C">
    <choice>
        <element name="E1" type="D1"/>
        <element name="E2" type="D2"/>
        . . .
    </choice>
</complexType>
```

The capacity of a <choice> complexType is the capacity of the child element with the greatest capacity:

capacity(*C*) = max(capacity(*E1*), capacity(*E2*), …)

4.3.9.3 Capacity of an Extension Complex Type

The value space created by extending another complex type is the product of the parent complex type that is being extended multiplied by the value space of the new extension content. Logarithmically, the capacity of the extended complex type is calculated by adding the capacity of the parent complex type to the capacity of the new extension content.

```
<complexType name="C">
    <complexContent>
        <extension base="P">
            C1
        </extension>
    <complexContent>
</complexType>
```

The capacity is the sum of the logarithmic capacity of the content and the logarithmic capacity of the parent type:

capacity(*C*) = capacity(*C1*) + capacity(*P*)

4.3.9.4 Capacity of a Restriction Complex Type

The value space created by restricting another complex type is equal to the value space of the new restricted content.

```
<complexType name="C">
    <complexContent>
        <restriction base="P">
            C1
        </restriction>
    <complexContent>
</complexType>
```

The capacity is the capacity of the restricted content:

capacity(*C*) = capacity(*C1*)

4.3.9.5 Capacity of a Complex Type with Attributes

The value space created by including one or more attributes in a complex type is the product of the value space of the complex type and each attribute. Logarithmically, the capacity is the sum of the capacity of the complex type and the capacity of each attribute.

```
<complexType name="C">
    C1
    <attribute name="A" type="D"/>
    . . .
</complexType>
```

The capacity is the sum of the logarithmic capacity of the complex type and the logarithmic capacity of each attribute:

capacity(*C*) = capacity(*C1*) + capacity(*A*) + …

4.3.10 Capacity of an Element with a Datatype That Is a Complex Type

The capacity of an element with a datatype that is a complex type is the capacity of the complexType.

<element name="*E*" type="*C*"/>

capacity(*E*) = capacity(*C*)

4.3.11 Capacity of a Namespace

The capacity of the elements in a namespace *N* is the maximum capacity of all globally declared elements (*E1*, *E2*, …) which are in *N*. Since a namespace may span multiple files, the capacity is the maximum capacity over all files.

capacity(*elements in namespace N*) = max(capacity(*E1*), capacity(*E2*), …)

The capacity of the attributes in a namespace *N* is the maximum capacity of all globally declared attributes (*A1*, *A2*, …) which belong to *N*. Since a namespace may span multiple files the capacity is the maximum capacity over all files.

capacity(*attributes in namespace N*) = max(capacity(*A1*), capacity(*A2*), …)

4.3.12 Capacity of the <any> Element

Case 1: The <any/> element

Since the <any/> element allows any element, from any namespace, no statement can be made regarding its capacity.

capacity(*any*) = infinite

Case 2: The <any namespace="*N*"/> element

The capacity of this element is the capacity of the elements in namespace *N*.

capacity(*any with namespace N*) = capacity(*elements in namespace N*)

4.3.13 Capacity of Fixed Values

Both elements and attributes may be declared to have a fixed (constant) value. Such elements/attributes have a capacity of zero bits.

<element name="*E*" fixed="*foo*"/>

capacity(*E*) = 0 bits

4.3.14 Capacity of the <anyAttribute> Element

Case 1: The <anyAttribute/> element

Since the <anyAttribute/> element allows any attribute to be present in an element, and an attribute may be of any simple datatype, the capacity can be as large as any simple datatype. Referring to Table 1, the capacity of any simple datatype is a function of the number of characters.

Let s = the number of characters

Cardinality: 128^s

capacity(*anyAttribute*) = 7s

Case 2: The <anyAttribute namespace="*N*"/> element

The capacity of this element is the capacity of the attributes in the namespace *N*.

capacity(*any with namespace N*) = capacity(*attributes in namespace N*)

4.3.15 Capacity of an XML Schema

An XML Schema can have several global elements, in which case there are several potential root elements for an XML instance document. Consequently, measuring the capacity of an XML Schema may involve generating a capacity number for each global element. This results in a set of capacity numbers, one for each global element. The capacity of an XML instance document is the capacity of the global element that is the root element of the XML instance document.

<element name="*E1*" type="*C1*"/>

<element name="*E2*" type="*C2*"/>

...

capacity(*schema*) = capacity(*E1*) or capacity(*E2*) or …

capacity(instance document with root element(*E1*) = capacity(*E1*)

Summary

This document describes how to measure the information capacity that is specified by an XML Schema. An XML Schema describes the format and content of XML instance documents. If a web service or XML application receives these instance documents, then in essence, the XML Schema describes the nature of the information channel to the web service or application.

This document treats an XML Schema as a specification of an information channel. It recommends how to determine the capacity of the information channel specified by an XML Schema.

The capacity is an indicator of how much effort will be required by a web service or XML application to process XML instance documents. Thus, it is an indicator of the potential load on the web service. Understanding the capacity of a channel specified by an XML Schema can be useful for comparing alternative designs. The capacity measure also can be used to improve web services or XML applications, and can help implement XML security capabilities.

Measuring the capacity of an XML Schema can be a useful tool to develop better ways to regulate or validate XML content, such as in a security system. A fundamental theorem in information theory is that the amount of regulation required will be at least as great as the amount of information received. Thus, XML instance documents that conform to an XML Schema that has a large capacity will require a correspondingly large effort to regulate or validate. A capacity measurement can provide developers of a security system with an indication of the feasibility or complexity associated with processing or validating XML instance documents. The measurement can help determine the suitability of using a specific XML Schema for enforcing format or content constraints on XML instance documents.

It might be possible to incorporate the capacity measurement into a risk assessment methodology that can help determine the probability of malicious or unauthorized data being present in an XML instance document. Additional research is appropriate to determine the effectiveness of capacity measurement in this manner.

References

[Ashby, 1961] *Introduction to Cybernetics* by Ross Ashby, 1961.

[Biron et al, 2004] *XML Schema Part 2: Datatypes*, Second Edition, World Wide Web Consortium, 28 October 2004. Document is available online at http://www.w3.org/TR/xmlschema-2/.

[Bishop, 2005] *Introduction to Computer Security* by Matt Bishop, 2005.

[Boyer et al, 2001] *Canonical XML, Version 1.0*, the World Wide Web Consortium, 15 March 2001. Document is available online at http://www.w3.org/TR/xml-c14n.

[Bray et al, 2004] *Extensible Markup Language* (XML), Version 1.1, World Wide Web Consortium Recommendation, February 2004 (edited 15 April 2004). Document is available online at http://www.w3.org/TR/xml11/.

[Fallside and Walmsley, 2004] *XML Schema Part 0: Primer,* Second Edition, World Wide Web Consortium, 28 October 2004. Document is available online at http://www.w3.org/TR/xmlschema-0/.

[Hull, 1986] *Relative Information Capacity of Simple Relational Database Schemata* by Richard Hull, Society for Industrial and Applied Mathematics Journal of Computing, 15(3):856-886, August 1986.

[Shannon, Weaver, 1949] *Mathematical Theory of Communication* by Claude Shannon and Warren Weaver, 1949.

[Thompson et al, 2004] *XML Schema Part 1: Structures,* Second Edition, World Wide Web Consortium, 28 October 2004. Document is available online at http://www.w3.org/TR/xmlschema-1/.

[Wheeler, 2003] *Validating Input* by David Wheeler, IBM, 23 October 2003. Document is available online at http://www-128.ibm.com/developerworks/linux/library/l-sp2.html.