HTTP Working Group
Internet Draft

D. Box
Developmentor

G. Kakivaya
A. Layman
S. Thatte
Microsoft
Corporation

D. Winer
Userland Software

Document: <draft-box-http-soap-00.txt>
Category: Informational

September 1999

SOAP: Simple Object Access Protocol

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026 except that the right to produce derivative works is not granted.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

1. Abstract

SOAP defines an RPC mechanism using XML for client-server interaction across a network by using the following mechanisms:

* HTTP as the base transport
* XML documents for encoding of invocation requests and responses

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [11].

3. Introduction

SOAP defines an "XML-RPC" protocol for client-server interaction across a network by using the following mechanisms:

* HTTP as the base transport
* XML documents for encoding of invocation requests and responses

SOAP is both low-entry and high-function, capable of use for simple stateless remote procedure calls as well as rich object systems.

SOAP works with today's deployed World Wide Web and provides extensibility mechanisms for future enhancements. For example, SOAP supports submitting invocations using both M-POST and POST.

3.1. Goals

* Provide a standard object invocation protocol built on Internet standards, using HTTP as the transport and XML for data encoding.
* Create an extensible protocol and payload format that can evolve over time.

3.2. Non-Goals

Define all aspects of a distributed object system, including the following:

* Distributed garbage collection
* Metadata discovery, type safety, and versioning
* Bi-directional HTTP communications
* Boxcarring or pipelining of messages
* Objects-by-reference (which requires distributed garbage collection and bi-directional HTTP)
* Activation (which requires objects-by-reference)

This specification lays the groundwork for a distributed object system. Getting consensus on a full object system would be a long and time-consuming process. Therefore, SOAP currently contains only the base features necessary to get the basic format and protocol working.

3.3. Examples of a SOAP Call

The call is to a StockQuote server, and the method is GetLastTradePrice. The method takes one string parameter, ticker, and returns a float.

3.3.1. Call

Following is an example of the SOAP encoding required to make this method call. This example uses the familiar HTTP verb POST. SOAP

mandates the use of the HTTP verb M-POST by preference over POST for reasons of extensibility and firewall friendliness. See section 6.1 for more information on M-POST.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml-SOAP
Content-Length: nnnn
MessageType: Call

 <GetLastTradePrice>
   <ticker>DIS</ticker>
 </GetLastTradePrice>
```

3.3.2. Response

Following is the return message containing the HTTP headers and XML body:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml-SOAP
Content-Length: nnnn
MessageType: CallResponse

<GetLastTradePriceResponse>
  <__return>34.5</__return>
</GetLastTradePriceResponse>
```

4. Relation to HTTP

In SOAP, the mechanism used for all communication is HTTP. (See [1].) Indeed, a central design goal of SOAP, perhaps the most important, is that SOAP be usable strictly on top of today's actually deployed World Wide Web infrastructure. That means SOAP has to live with and work in the face of various levels of HTTP implementation, the active use of firewalls and proxies, and so on. Some aspects of SOAP, such as the permitted use of HTTP methods

beyond those of classic HTTP, are designed to anticipate, and thus make use of, some evolution and improvement in this base, but nothing in SOAP can require such fundamental changes in order for SOAP to function.

SOAP defines a new Content-Type of "text/xml-SOAP". This is used to specify the body of the HTTP message containing a XML encoded method call.

To disambiguate the headers it adds to HTTP, SOAP makes use of the HTTP Extension Framework specification (See [2]). To facilitate firewall filtering, SOAP adds new headers to HTTP.

Unless otherwise indicated in this document, existing practices with respect to the handling of HTTP requests and responses are to be adhered to. Specifically, this includes the following:

* Redirection
* Caching
* Connection management
* Support for access authentication and security

5. Relation to XML

XML is used to encode the call and response bodies. See [3] for more information on XML.

All protocol tags may be scoped to the SOAP namespace. Use of namespaces in SOAP is optional. The SOAP namespace has the proposed value "http://w3.org/Schemas/SOAP/kw". See [6] for more information on XML namespaces.

No XML document forming the HTTP request of a SOAP invocation may require the use of an XML DTD in any manner.

SOAP uses the ID attribute "id" to specify the unique identifier of an encoded element. SOAP uses the attribute "href" to specify a reference to that value, in a manner conforming to the XML Linking Language specification working draft. See [9] for more information on XML Linking Language.

It is worth noting that the rules governing XML payload format in SOAP are entirely independent of the fact that the payload is carried over an HTTP transport.

6. Method Invocation

A method invocation is performed by creating the HTTP request header and body and processing the returned response header and body. The request and response headers consist of standard and extended HTTP headers.

The following sections will cover the use of standard HTTP headers and the definition of extended HTTP headers.

6.1. HTTP Verb Rules

SOAP allows two verb options within the Call HTTP header: M-POST or POST.

The verb M-POST is an extension verb based on in the HTTP Extension Framework specification. (See [2].) A SOAP invocation must first try the invocation by using M-POST.

If the M-POST invocation fails, it must retry using the HTTP method POST. The details of this mechanism are provided below. The purpose of supporting this extended invocation mechanism in SOAP is to provide a mechanism to unambiguously add headers to the HTTP protocol.

6.2. Using M-POST vs. POST

Since a design goal of the use of M-POST is to provide Internet firewalls and proxies greater administrative flexibility, careful attention must be paid as to when a SOAP client uses the M-POST method vs. the POST method. The rules are as follows:

When carrying out an invocation, a SOAP client must first try the invocation using the M-POST invocation style.

If that M-POST invocation fails with an HTTP status of "501 Not Implemented" or "510 Not Extended," the client should retry the request using the POST invocation style. If that POST invocation fails with an HTTP status of "405 Method Not Allowed," the client should fail the request. If any other HTTP error is returned, it should be processed according to the HTTP specification.

Further, if such a failure code is received on an M-POST, then in subsequent invocations to the same HTTP server, the client may omit

the attempt at M-POST invocations for a period of 24 hours, thereby avoiding the need for an extra round-trip on each and every method invocation.

Given this algorithm, firewalls can effectively force the use of M-POST for SOAP invocations by prohibiting POST invocations of Content-Type "text/xml-SOAP".

6.3. Method Invocation HTTP Headers

M-POST and POST messages on call or response must include a header "MessageType" whose value is either "Call" or "CallResponse," to indicate the type of message in the payload.

The payload and Content-Type of a method call are identical to a method response except in the following circumstances:

* The method call must contain additional HTTP header fields in the request:

a) If using the M-POST verb, a mandatory extension declaration must be present that refers to the namespace "http://www.microsoft.com/protocols/ext/SOAP". For the purposes of this section, suppose that said declaration chooses to map the namespace to the header-prefix "01". If the POST verb is used, the namespace header-prefix is not used. For example, a MethodName header would have an M-POST value of "01-MethodName" and a POST value of "MethodName".

b) If an interface name is necessary to perform the invocation, the request must include a header "InterfaceName" whose value is the interface on the server. If interfaceName is not required in the method invocation, an "InterfaceName" header must not be present.

c) If a method name is necessary to perform the invocation, then the request MUST include a header "MethodName" whose value is the method to be invoked on the target. If method name is NOT required in the method invocation, then a "MethodName" header MUST NOT be present

* The server must fail the request if the required headers are missing. The failure HTTP response status-line should contain the value "400", which means "Bad Request".

6.4. Method Invocation Body

A SOAP method invocation consists of a method call and optionally a method response. The method call and method response body consists of an HTTP header and the XML payload. The XML payload consists of the root, call, and response elements, and, optionally, header information.

The body components are defined as follows:

* The SOAP root element is the top element in the XML tree.

* The SOAP payload headers contain implicit information that needs to travel with the call.

* The call is the encoded call with parameters that is passed to the server. It is a child of the root element.

* The response is the return value or error/exception that is passed back to the client. It is a child of the root element.

The encoding rules are as follows:

1) Root element
a) The element tag is "SerializedStream". The root element provides serialization scope and as such is optional when it has a single child element.
b) It may contain an attribute "main" whose value is a URI fragment identifier to the call or response element. If the attribute "main" is absent, the call or response element must be the first element scoped within the serialized stream.
c) It may contain an attribute "headers" whose value is a URI fragment identifier to the headers element. All the elements referenced directly or indirectly by the header element must always precede those reachable directly or indirectly from the call or response element.

d) It may contain an attribute "serializationPattern" that indicates any serialization rules used in addition to those required by the SOAP spec.
e) It may contain namespace declarations.
f) It may contain additional attributes, provided these are namespace-qualified.

2) SOAP payload headers
a) The element tag is "headers".
b) It must contain an "id" attribute that the root element references.

c) It contains a list of header entries.
d) Standard entries include the following:
i) "MethodSig" that contains an implementation-specific value used to disambiguate overloaded methods.
ii) "InterfaceName" that contains the interface to invoke on.
iii) "UnorderedParams" that contains a Boolean indicating whether or not the parameters are unordered. The default is for parameters to be ordered.

3) Call
a) The element tag is the method name.
b) It may contain an "id" attribute that the root element references.
c) It contains child elements for each [in] and [in/out] parameter. The element names are the parameter names or "__param" prefixed to the ordinal-representing position of the parameter starting at 0.
d) It may contain a "version" attribute that specifies the version of the call object.

4) Response
a) The element tag is "Response" appended to the method name.
b) It may contain an element "__return" containing the return value.
c) It contains child elements for each [in/out] and [out] parameter. The element names are the parameter names or "__param" prefixed to the ordinal-representing position of the parameter starting at 0.
d) It may contain an element "__fault" if an error occurred. When a "__fault" element is present, elements mentioned in b) and c) must not be present.
e) It may contain a "version" attribute that specifies the version of the response object.

If the call or response version attribute is not specified, the default value of "1.0" is used. A server must use the version passed in the call for encoding the response, or it must fail the request. In the case where the server accepts a version or level less than its maximum, it must respond to the client by using the same version and level. If a server receives a version it cannot handle, the HTTP response status-line should contain the value "400", which means

"Bad Request", and contain a fault in the call response with the fault code SOAP_E_VERSION_MISMATCH.

See section 7 for information on how to encode parameter values.

6.5. SOAP Payload Headers

In addition to the elements that specify direct, explicit information about the call or response, SOAP provides a way to pass extended, implicit information with the call through the use of the "headers" element. It is referenced by and encoded as a child of the root XML element. It contains a collection of distinctly named entries.

An example of the use of the header element is the passing of an implicit transaction ID along with a call. Since the transaction ID is not part of the signature and is typically held in an infrastructure component rather than application code, there is no direct way to pass the necessary information with the call. By adding an entry to the headers and giving it a fixed name, the transaction manager on the receiving side can extract the transaction ID and use it without affecting the coding of remote procedure calls.

Each header entry is encoded as an embedded element. The encoding rules for a header are as follows:

1. The element's name identifies the header.
2. The element may contain an attribute "href" that refers to the header's value if the value is independently encoded. (See Section 6 for details on encoding terms and rules.)
3. If the element does not contain an attribute "href" referring to a value, the elment must contain an attribute "type" specifying the type of the immediately contained value.
4. The element may contain an attribute "mustUnderstand" specifying required understanding of the header by the destination.

An example is a header with an identifier of "TransactionID", a "mustUnderstand" value of true, and an integer value of 5. This would be encoded as follows:

```
<TransactionID type="int" mustUnderstand="1">5</TransactionId>
```

6.5.1. The "mustUnderstand" Attribute

Header entries may have an attribute "mustUnderstand". This may have one of two values, either "1" or "0". The absence of such a "mustUnderstand" attribute is semantically equivalent to its presence with the value "0".

If a header element is tagged with a "mustUnderstand" with value "1", a SOAP implementation processing the element must understand the semantics intended for the element (as conveyed by its element tag, contextual setting, and so on) and honor those semantics. If the SOAP implementation doesn't understand the element, it must return an error as specified in section [5.1], "Results from a Method Call."

The idea is to allow for robust semantic extensibility and change. Headers tagged with mustUnderstand="1" can be presumed to somehow concretely change or modify the semantics of their element. Tagging the headers in this manner assures that this change in semantics will not be silently (and, presumably, erroneously) ignored by those who may not fully understand it.

If the "mustUnderstand" field is missing or has a value of "0", that element can safely be ignored.

For example: If the client passed along a transaction ID header, as in the above example, with a "mustUnderstand" of "1", then the server should fail if it cannot process the transaction ID and comply with the transactional semantics.

6.6. Making a Method Call

To make a method call, the following information is needed:

* The URI of the target objec,
* An optional interface nam,
* A method name
* An optional method signature
* The parameters to the method
* Optional header data

The target URI of the HTTP request indicates the resource that the invocation is being made against; in this specification, we refer to that resource as the "server address," to distinguish it from other uses of URIs. Other than it be a valid URI, SOAP places no restriction on the form of an address. See [8] for more information on URIs.

The body of a SOAP method call must be of Content-Type 'text/xml-SOAP'.

The SOAP protocol places no absolute restriction on the syntax or case-sensitivity of interface names, method names, or parameter names. Of course, individual SOAP servers will respond to only the names they support; the selection of these is at their own sole

discretion. The one restriction is that the server must preserve the case of names.

6.6.1. Representation of Method Parameters

Method parameters are encoded as child elements of the call or response, encoded using the following rules:

1) The name of the parameter in the method signature is used as the name of the corresponding element.
2) The parameter elements may contain a "type" attribute.
3) Parameter values are expressed using the rules in section 6 of this document.

6.6.2. Sample Encoding

This sample is the same call as in section 3.3.1 but uses optional headers and no parameter names. It uses XML namespaces to disambiguate SOAP keywords. The call element is not the first element nested within the root and is referenced by a main attribute in the root element.

```
<SerializedStream headers="ref-0" main="ref-1"
xmlns:SOAP="http://w3.org/Schemas/SOAP/kw"
serializationPattern="urn:schemas-microsoft-com:soap.v1">
     <SOAP:headers id="ref-0">
             <TransactionId type="int"
mustUnderstand="1">5</TransactionId>
     </SOAP:headers>
     <GetLastTradePrice id="ref-1">
             <__param0 id="ref-2">DIS</__param>
     </GetLastTradePrice>
</SerializedStream>
```

7. Results of Method Calls

At the receiving site, a call request can have one of the following four outcomes:

a) The HTTP infrastructure on the receiving site was able to receive and process the request.

b) The HTTP infrastructure on the receiving site could not receive and process the request.

c) The SOAP infrastructure on the receiving site was able to decode the input parameters, dispatch to an appropriate server indicated by the server address, and invoke an application-level function corresponding semantically to the interface or method indicated in the method call.

d) The SOAP infrastructure on the receiving site could not decode the input parameters, dispatch to an appropriate server indicated by the server address, and invoke an application-level function corresponding semantically to the interface or method indicated in the method call.

In the first case, the HTTP infrastructure passes the headers and body to the SOAP ifrastructure.

In the second case, the result is an HTTP response containing an HTTP error in the status field and no XML body.

In the third case, the result of the method call consists of a result message.

In the fourth case, the result of the method is a fault message indicating a fault that prevented the dispatching infrastructure on the receiving side from successful completion.

In the third and fourth cases, additional payload headers may for extensibility again be present in the results of the call.

7.1. Results from a Method Call

The results of the call are to be provided in the form of a call response. The HTTP response must be of Content-Type "text/xml-SOAP".

Because a result indicates success and a fault indicates failure, it is an error for the method response to contain both a result and a fault.

7.2. SOAPFault and HTTP Status Codes

If the HTTP infrastructure successfully processes the Call, passes it to the SOAP infrastructure, and an error occurs, an exception is passed to the caller in the fault element of the response. That exception can contain any record or structure. In this section, a simple exception record is defined. This record must be supported by the SOAP infrastructure and is used to return errors in the SOAP

infrastructure.

```
struct SOAPFault
{
     int             faultcode;
     String  faultstring;
     int             runcode;
}
```

Three members of this structure are defined, as follows:

* "faultcode", which must contain a numeric value. The value should be taken from the space of SOAP status codes, described below. The faultcode is intended for use by software.

* "faultstring", which must contain a string value. The faultstring is intended for use by human users and must not be acted upon algorithmically by software. faultstring is similar to the

'Reason-Phrase' that may be present in HTTP responses. (See [1], section 6.1.)

* "runcode", which must contain a numeric value. The runcode is intended to indicate whether or not the request reached the destination server. There are three runcodes currently defined: 0 - Maybe, 1 - No, 2 - Yes.

Other struct members beyond the three described above may be present.

If the fault specifies a server fault, as opposed to an HTTP fault, the HTTP status code must be "200" and the HTTP status message must be "OK". If it specifies an HTTP fault, the HTTP status code as defined in the HTTP specification [1] should be used.

If a method call fails to be processed because of a non-understood extension header element contained therein, the method invocation must return a SOAPFault. The SOAPFault must contain a 'faultcode' of SOAP_E_MUSTUNDERSTAND.

If a method response fails to be processed for similar reasons, an appropriate exceptional condition should be indicated to the application layer in an implementation-defined manner.

7.3. SOAP Status Codes

SOAP defines its own space of numeric status codes. This space is used only by the SOAP infrastructure and is not expected to be used on HTTP failure. The reason this space is defined is to aid the conversion of existing protocols onto SOAP.

This status code space must be used for faultcodes contained in SOAPFaults and in the method definitions defined in this specification that return status code values. Further, use of this space is recommended (but not required) in the specification of methods defined outside of the present specification.

The SOAP status code space contains numeric values drawn from the following ranges:

a) The HTTP Status Code Definitions, defined in Section 10 of RFC2068. (See [1].) Such values are three-digit numbers in the range 100-999 (decimal).

b) 0x8011FE00-0x8011FFFF (decimal: 2,148,662,784 - 2,148,663,295)

c) 0x0011FE00-0x0011FFFF (decimal: 1,179,136 - 1,179,647)

This specification at present defines the following status codes beyond those specified in [1]:

Name	Value	Meaning
SOAP_E_VERSION_MISMATCH	0x8011FE00	The call was using an unsupported SOAP version.
SOAP_E_MUSTUNDERSTAND	0x8011FE01	An XML element was received that contained an element tagged with mustUnderstand="1" that was not understood by the receiver.

7.4. Examples of Response Messages

The response from the example in section 3.3.2 would be:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml-SOAP
Content-Length: nnnn
MessageType: CallResponse

<SerializedStream
```

```
serializationPattern="urn:schemas-microsoft-com:soap.v1">
<GetLastTradePriceResponse>
     <__return>34.5</__return>
</GetLastTradePriceResponse>
</SerializedStream>
```

If there was an error in the HTTP infrastructure, the response could be as follows:

```
HTTP/1.1 401 Unauthorized
Connection: close
```

If there was an error in the SOAP infrastructure processing the request on the server, the response could be as follows:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml-SOAP
Content-Length: nnnn
MessageType: CallResponse

<SerializedStream
serializationPattern="urn:schemas-microsoft-com:soap.v1">
<GetLastTradePriceResponse>
     <__fault>
             <faultcode>0x8011FE00</faultcode>
             <faultstring id="ref-2">SOAP Must Understand
Error</faultstring>
             <runcode>1</runcode>
     </__fault>
</GetLastTradePriceResponse>
</SerializedStream>
```


If the application passed back its own exception, the request response would be as follows:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml-SOAP
Content-Length: nnnn
MessageType: CallResponse

<SerializedStream main="#ref-0"
serializationPattern="urn:schemas-microsoft-com:soap.v1">
<GetLastTradePriceResponse id="ref-0">
```

```
     <__fault href="#ref-1"/>
</GetLastTradePriceResponse>
<MyExceptionType href="#ref-1">
     <message type="string">My application didn't work</message>
     <errorcode type="int">1001</errorcode>
</MyExceptionType>
</SerializedStream>
```

8. Types

SOAP uses a simple, traditional type system. A type either is a simple (scalar) type or is a compound type constructed as a composite of several parts, each with a type.

Because all types are contained or referenced within a call or response element, the encoding samples in this section assume all namespace declarations are at a higher element level.

8.1. Rules for Encoding Types in XML

XML allows very flexible encoding of data to represent a method call. SOAP defines a narrower set of rules for encoding. This section defines the encoding rules at a high level, and the next section describes the encoding rules for specific types when they require more detail.

To describe encoding, the following terminology is used:

1. A "type" includes integer, string, point, or street address. A type in SOAP corresponds to a scalar or structured type in a programming language or database. All values are of specific types.
2. A "compound type" is one that has distinct, named parts and whose encoding should reflect those named parts. A "simple type" is one without named parts. A structured type in a programming language is a compound type, and so is an array.
3. The name of a parameter or of a named part of a compound type is called an "accessor."
4. If only one accessor can reference it, a value is considered "single-reference" for a given schema. If referenced by more than one, actually or potentially, it is "multi-reference." Therefore, it

is possible for a certain type to be considered "single-reference" in one schema and "multi-reference" in another schema.
5. Syntactically, an element may be "independent" or "embedded." An independent element is contained immediately by the root element. An embedded element is contained within a non-root element.

The rules are as follows:

1. Elements may be used to reflect either accessors or instances of types. Embedded elements always reflect accessors. Independent elements always reflect instances of types. When reflecting an accessor, the name of the element gives the name of the accessor. When reflecting an instance of a type, the name of the element gives the name of the type.
2. A call or response is always encoded as an independent element.
3. Accessors are always encoded as embedded elements.
4. A value (simple or compound) is encoded as element content, either of an element reflecting an accessor to the value or of an element reflecting an instance of that type.
5. A simple value is encoded as character data--that is, without any subelements.
6. Strings and byte arrays are multi-reference simple types, but special rules allow them to be represented efficiently for common cases. An accessor to a string or byte-array value may have an attribute named "id" and of type "ID" per the XML Specifications. If so, all other accessors to the same value are encoded as empty elements having an attribute named "href" and of type "URI" per the XML Linking Language Specifications, with the href containing a URI fragment identifier referencing the single element containing the value.
7. It is permissible to encode several references to a simple value as though these were references to several single-reference values, but only when from context it is known that the meaning of the XML instance is unaltered.
8. A compound value is encoded as a sequence of elements, each named according to the accessor it reflects. (See also section 8.4.1.)
9. A multi-reference simple or compound value is encoded as an independent element containing an attribute named "id" and of type "ID" per the XML Specifications. Each accessor to this value is an empty element having an attribute named "href" and of type "URI" per the XML Linking Language Specifications, with the href containing a URI fragment identifier referencing the corresponding independent element.
10. Arrays are compound types. Arrays can be of one or more dimensions(rank) whose elements are normally laid contiguously in memory. Arrays can be single-reference or multi-reference values. Single-reference embedded arrays are encoded using accessor elements. Multi-reference arrays are encoded as independent elements named "Array". The independent element or the accessor must contain a "type" attribute that specifies the type and dimensions of the array and is encoded as the type of the array element, followed by "[", followed by comma-separated lengths of each dimension, followed by "]". Note that the array element itself can be an array. An array

type is encoded as its element type, followed by "[", followed by rank encoded as a sequence of commas(one for each dimension), followed by "]". It may also contain an "offset" attribute to indicate the starting position of a partially represented array. Each element of an array is encoded using the accessor named "item". The elements are represented as a list with the dimension on the right side varying rapidly. The "item" accessor may contain the "position" attribute that conveys the position of the item in the enclosing array. Both "offset" and "Position" attributes are encoded as "[", followed by a comma-separated position in each dimension, followed by "]".

11. Any accessor element that contains its value directly may optionally have an attribute named "type" whose value indicates the type of the element's contained value.

12. A NULL reference is encoded as an independent element named SOAPNULL containing an attribute named "id" and of type "ID" per the XML Specifications. Each accessor to this value is an empty element having an attribute named "href" and of type "URI" per the XML Linking Language Specifications, with the href containing a URI fragment identifier referencing the SOAPNULL independent element.

8.2. Simple Types

For simple types, SOAP adopts the types found in the section "Specific Datatypes" of the XML-Data Specification (see [4]), along with the corresponding recommended representation thereof. Examples include:

```
ui4:           58502
 float:          .314159265358979E+1
 i2:           -32768
```

Strings and arrays of bytes are encoded as multi-reference simple types.

8.2.1. String

A string is a multi-reference simple type. According to the rules of multi-reference simple types, the containing element of the string value may have an ID attribute; additional accessor elements may then have matching href attributes.

For example, two accessors to the same string could appear, as follows:

```
<greeting id="String-0">Hello</greeting>
<salutation href="#String-0"/>
```

However, if the fact that both accessors reference the same instance of the string is immaterial, they may be encoded as though single-

reference, as follows:

<greeting>Hello</greeting>

<salutation>Hello</salutation>

8.2.2. Array of Bytes

An array of bytes is encoded as a multi-reference simple type. The recommended representation of an opaque array of bytes is the 'bin.base64' encoding defined in XML DCD (see [5]), which simply references the MIME standard. However, the line length restrictions that normally apply to Base64 data in MIME do not apply in SOAP.

bin.base64: aG93IG5vdyBicm93biBjb3cNCg==

8.3. Variant

Many languages allow accessors that can polymorphically access values of several types, each type being available at run-time. When the value is single-reference, the type of this kind of accessor is often called "Variant". A Variant accessor must contain a "type" attribute that describes the type of the actual value.

For example, a Variant parameter named "cost" with a type of float would be encoded as follows:

<cost type="float">29.95</cost>

as constrasted with a cost parameter whose type is invariant, as follows:

<cost>29.95</cost>

8.4. Compound Types

Beyond the simple types, SOAP defines support for the following constructed types:

* Records/structs
* arrays

Where appropriate and possible, the representation in SOAP of a value of a given type mirrors that used by practitioners of XML-Data and the common practice of the XML community at large.

8.4.1. Compound Values and References to Values

A compound value contains an ordered sequence of structural members. When the members have distinct names, as in an instance of a C or C++ "struct", this is called a "struct," and when the members do not have distinct names but instead are known by their ordinal position, this is called an "array..

The members of a compound value are encoded as accessor elements. For a struct, the accessor element name is the member name. For an array, the accessor element name is "item" and the sequence of the accessor elements follows the ordinal sequence of the members.

The following is an example of a struct of type Book:

```
<Book>
<author>Henry Ford</author>
<preface>Prefatory text</preface>
<intro>This is a book.</intro>
</Book>
```

Below is an example of a type with both simple and compound members. It shows two levels of referencing.

Note that the "href" attribute of the Author accessor element is a reference to the value whose "id" attribute matches; a similar construction appears for the Address.

```
<Book>
<title >My Life and Work</title>
<author href="#Person-1"/>
</Book>
<Person id="Person-1">
<name>Henry Ford</name>
<address href="#Address-2"/>
</Person>
<Address id="Address-2">
<email>henryford@hotmail.com</email>
<web>www.henryford.com</web>
</Address>
```

The form above is appropriate when the Person value and the Address value are multi-reference. If these were instead both single-reference, they would not need to be independent but could instead be embedded, as follows:

```
<Book>
<title>My Life and Work</title>
<author>
 <name>Henry Ford</name>
 <address>
  <email>henryford@hotmail.com</email>
  <web>www.henryford.com</web>
 </address>
</author>
</Book>
```

If instead there existed a restriction that no two persons can have the same address in a given schema and that an address can be either a Street-address or an Electronic-address, a Book with two authors would be encoded in such a schema as follows:


```
<Book>
<title >My Life and Work</title>
<firstauthor href="#Person-1"/>
<secondauthor href="#Person-2"/>
</Book>
<Person id="Person-1">
<name>Henry Ford</name>
<address type="Electronic-address">
 <email>henryford@hotmail.com</email>
 <web>www.henryford.com</web>
</address>
</Person>
<Person id="Person-2">
<name>Thomas Cook</name>
<address type="Street-address">
 <Street>Martin Luther King Rd</Street>
 <City>Raliegh</City>
<State>North Carolina</State>
</address>
</Person>
```

8.4.1.1. Generic Records

There are cases where a struct is represented with its members named and values typed at run time. Even in these cases, the existing rules apply. Each member is encoded as an element with matching name, and each value is either contained or referenced. Contained values must have a "type" attribute giving the type of the value.

8.4.2. Arrays

The representation of the value of an array is an ordered sequence of elements constituting items of the array. Each element is named "item".

As with compound types generally, if the type of an item in the array is a single-reference type, each item contains its value. Otherwise, the item references its value via an href attribute.

The following example is an array containing integer array members. The length attribute is optional.

```
 <Array type="int[2]">
   <item>3</item>
   <item>4</item>
 </Array>
```

The following example is an array of Variants containing an integer and a string.

```
 <Array type="variant[2]">
   <item type="int">23</item>
```



```
   <item type="string" id="ref-0">some silly old string</item>
 </Array>
```

The following is an example of a two-dimensional array of strings.

```
 <Array type="string[3,3]">
   <item>r1c1</item>
   <item>r1c2</item>
   <item>r1c3</item>
   <item>r2c1</item>
   <item>r2c2</item>
   <item>r2c3</item>
 </Array>
```

The following is an example of an array of two arrays, each of which is an array of strings.

```
 <Array type="string[][2]">
     <item href="#array-1"/>
     <item href="#array-2"/>
</Array>
<Array id="array-1" type="string[3]">
<item>r1c1</item>
```

```
<item>r1c2</item>
<item>r1c3</item>
</Array>
 <Array id="array-2" type="string[2]">
<item>r2c1</item>
<item>r2c2</item>
</Array>
```

Finally, the following is an example of an array of phone numbers embedded in a struct of type Person:

```
<Person>
<name>John Hancock</name>
     <phone-numbers type="string[2]">
     <item>111-2222</item>
     <item>999-0000</item>
</phone-numbers>
</Person>
```

8.4.2.1. Partially transmitted arrays

SOAP provides support for partially transmitted arrays, known as "varying" arrays, in some contexts. (See [7].) A partially transmitted array indicates in an "offset" attribute the zero-origin index of the first element transmitted; if omitted, the offset is taken as zero.

The following is an example of an array of size five that transmits only the third and fourth element:


```
 <Array type="string[5]" offset="[2]">
  <item>The third element</item>
  <item>The fourth element</item>
 </Array>
```

8.4.2.2. Sparse Arrays

SOAP provides support for sparse arrays in some contexts. Each element contains a "position" attribute that indicates its position within the array. The following is an example of array of arrays of strings:

```
 <Array type="string[,][2]">
     <item href="#array-1" position="[2]"/>
</Array>
```

```
 <Array id="array-1" type="string[10,10]">
     <item position="[2,2]">The second element"</item>
     <item position="[7,2]">The seventh element</item>
 </Array>
```

Assuming that the only reference to array-1 occurs in the enclosing array, this example could also have been encoded as follows:

```
 <Array type="string[,][2]">
<item position="[2]">
 <Array type="string[10,10]">
     <item position="[2,2]">The second element"</item>
     <item position="[7,2]">The seventh element</item>
 </Array>
</item>
</Array>
```

8.5. Default Values

An omitted accessor element implies either a default value or that no value is known. The specifics depend on the accessor, method, and its context. Typically, an omitted accessor implies a Null value for Variant and for polymorphic accessors (with the exact meaning of Null accessor-dependent). Typically, an omitted Boolean accessor implies either a False value or that no value is known, and an omitted numeric accessor implies either that the value is zero or that no value is known.

9. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [10].

10. Security Considerations

Not described in this document are methods for integrity and privacy protection. Such issues will be addressed more fully in a future version(s) of this document.

11. References

[1] RFC2068: Hypertext Transfer Protocol, http://info.internet.isi.edu/in-notes/rfc/files/rfc2068.txt. Also:

http://www.w3.org/Protocols/History.html.

[2] HTTP Extension Framework, http://www.w3.org/Protocols/HTTP/ietf-http-ext.
[3] The XML Specification, http://www.w3.org/TR/WD-xml-lang.
[4] XML-Data Specification, http://www.w3.org/TR/1998/NOTE-XML-data.
[5] Document Content Description for XML, http://www.w3.ort/TR/NOTE-dcd.
[6] Namespaces in XML, http://www.w3.org/TR/REC-xml-names.
[7] Transfer Syntax NDR, in "DCE 1.1: Remote Procedure Call," http://www.rdg.opengroup.org/onlinepubs/9629399/toc.htm.
[8] RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax and Semantics, http://www.ietf.org/rfc/rfc2396.txt.
[9] XML Linking Language, http://www.w3.org/1999/07/WD-xlink-19990726.
[10] RFC-2234: Augmented BNF for Syntax Specifications: ABNF
[11] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

12. Author's Addresses

G. Kavivaya
Microsoft
One Microsoft Way
Redmond, WA 98052
Email: gopalk@microsoft.com

Full Copyright Statement

"Copyright (C) The Internet Society (date). All Rights Reserved. This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implmentation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into