Web Services Policy Framework (WS-Policy)

Version 1.01
21 March 2003

Authors

Don Box, Microsoft
Francisco Curbera, IBM
Maryann Hondo (Editor), IBM
Chris Kaler (Editor), Microsoft
Dave Langworthy, Microsoft
Anthony Nadalin, IBM
Nataraj Nagaratnam, IBM
Mark Nottingham, BEA
Claus von Riegen, SAP
John Shewchuk, Microsoft

Copyright Notice

Abstract

The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service.

WS-Policy defines a base set of constructs that can be used and extended by other Web Services specifications to describe a broad range of service requirements, preferences, and capabilities.

Composable Architecture

The Web service specifications (WS*) are designed to be composed with each other to provide a rich set of tools to provide security in the Web Services environment. WS-Policy by itself does not provide a negotiation solution for Web services. WS-Policy is a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of policy exchange models.

Status

This WS-Policy Specification is an initial public draft release and is provided for review and evaluation only. BEA, IBM, Microsoft, and SAP hope to solicit your contributions and suggestions in the near future. BEA, IBM Microsoft, and SAP make no warrantees or representations regarding the specifications in any manner whatsoever.

Table of Contents

1. Introduction

WS-Policy provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web Services-based system. WS-Policy defines a framework and a model for the expression of these properties as policies. Policy expressions allow for both simple declarative assertions as well as more sophisticated conditional assertions.

WS-Policy defines a policy statement to be a collection of one or more policy assertions. Some assertions specify traditional requirements and capabilities that will ultimately manifest on the wire (e.g., authentication scheme, transport protocol selection). Some assertions specify requirements and capabilities that have no wire manifestation yet are critical to proper service selection and usage (e.g., privacy policy, QoS characteristics). WS-Policy provides a single policy grammar to allow both kinds of assertions to be reasoned about in a consistent manner.

WS-Policy stops short of specifying how policy statements are discovered or attached to a Web service. Other specifications are free to define technology-specific mechanisms for associating policy with various entities and resources. Subsequent specifications will provide profiles on WS-Policy usage within other common Web service technologies.

1.1 Goals

The goal of WS-Policy is to provide the mechanisms needed to enable Web Services applications to specify policy information. Specifically, this specification defines the following:

- An XML-based structure called a policy expression which contains domain specific Web Service policy information
- A core set of grammar elements to indicate how the contained policy assertions apply.

1.2 Example

The following example illustrates a policy:

```
001 <wsp:Policy xmlns:wsse="..." xmlns:wsp="...">
002   <wsp:ExactlyOne>
003     <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="100">
004       <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
005     </wsse:SecurityToken>
006     <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="1">
007       <wsse:TokenType>wsse:X509v3</wsse:TokenType>
008     </wsse:SecurityToken>
009   </wsp:ExactlyOne>
010 </wsp:Policy>
```

In this example, we illustrate the expression of a security policy using assertions defined in WS-SecurityPolicy [WS-SecurityPolicy]. Lines 1 to 10 represent policy statements for authentication.

Lines 2 to 9 represent the <wsp:ExactlyOne> policy operator that is used to group policy assertions into policy statements. That is, a valid policy can contain any one of the contained assertions (lines 3 to 8).

Lines 3 to 5 and 6 to 8 represent two specific security policy assertions that indicate that two types of authentication are supported and that of the two types, Kerberos authentication is preferred over X509 authentication.

2. Notations and Terminology

This section specifies the notations, namespaces, and terminology used in this specification.

2.1 Notational Conventions

The keywords “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 [RFC 2119].

Namespace [XML-NS] URIs (of the general form “some-URI”) represents some application-dependent or context-dependent URI as defined in RFC 2396 [RFC 2396].

WS-Policy is designed to work with the general Web Services framework including WSDL service descriptions [WSDL], UDDI businessServices and bindingTemplates [UDDI], and SOAP message structure and message processing model [SOAP11, SOAP12]. WS-Policy should be applicable to any version of SOAP; the current SOAP 1.2 namespace URI is used herein to provide detailed examples, but there is no intention to limit the applicability of this specification to a single version of SOAP.

2.2 Namespaces

The XML namespace URI that MUST be used by implementations of this specification is:

```
http://schemas.xmlsoap.org/ws/2002/12/policy
```

The following namespaces are used in this document:

Prefix	Namespace
S	http://www.w3.org/2002/06/soap-envelope
wsdl	http://schemas.xmlsoap.org/wsdl/
wsse	http://schemas.xmlsoap.org/ws/2002/12/secext
wsp	http://schemas.xmlsoap.org/ws/2002/12/policy
wsu	http://schemas.xmlsoap.org/ws/2002/07/utility
xsd	http://www.w3.org/2001/XMLSchema

In this document reference is made to the *wsu:Id* attribute in a utility schema (http://schemas.xmlsoap.org/ws/2002/07/utility). The *wsu:Id* attribute is included in the utility schema with the intent that other specifications requiring a global Id could reference it (as is done here).

2.3 Terminology

We introduce the following terms which are used throughout this document:

Policy – A *policy* is an informal abstraction and a term that is often used to refer to the set of information that is being expressed as policy assertions and policy statements.

Policy Statement – A *policy statement* is a group of policy assertions.

Policy Assertion – A *policy assertion* represents an individual preference, requirement, capability or other property.

Policy Expression – A *policy expression* is an XML Infoset representation of one or more policy statements.

Policy Subject – A policy subject is an entity (e.g., an endpoint, object, or resource) to which a policy expression can be bound.

Policy Attachment – The mechanism for associating policy expressions with one or more subjects is referred to as *policy attachment*.

3. Policy Model

This section defines an abstract model for policies and for operations upon policies.

This abstract model is independent of a serialized representation of the model.

At the abstract level a policy is the set of conditions on an action that may result in a behavior that reflects these conditions. In typical applications, there are many types of policies expressed at different levels of detail.

For Web services, the provider of a Web service generally has conditions under which it provides the service. A requester might use this information to decide whether or not to use the service. Sometimes the provider of the service allows the requester to further qualify an expressed policy.

To make this abstract model concrete for interoperability, the next section defines an XML-based grammar for expressing a Web service's policy. Subsequent specifications will define how policy statements are associated with Web Service elements as well as how domain specific policy assertions are expressed.

3.1 Policy Expressions

Any policy can be represented by a policy expression. Policy expressions are a compact and interoperable means to author and convey policy statements. It should be noted that this specification does not define how to specify the subject of a policy expression.

A policy expression is an XML serialization (an Infoset) that adheres to the syntax described by this section. That grammar consists of three basic components: the top level <wsp:Policy> container element, the policy operators (which provide a way to group assertions), and several common attributes to distinguish usage. All three are shown in the example below (using custom assertions invented for this example):

```
001 <wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
002   <wsp:ExactlyOne>
003     <wsp:All wsp:Preference="100">
004       <wsse:SecurityToken>
005         <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
006       </wsse:SecurityToken>
```

```
007       <wsse:Integrity>
008         <wsse:Algorithm Type="wsse:AlgSignature"
                  URI="http://www.w3.org/2000/09/xmlenc#aes" />
009       </wsse:Integrity>
010     </wsp:All>
011     <wsp:All wsp:Preference="1">
012       <wsse:SecurityToken>
013         <wsse:TokenType>wsse:X509v3</wsse:TokenType>
014       </wsse:SecurityToken>
015       <wsse:Integrity>
016         <wsse:Algorithm Type="wsse:AlgEncryption"
                  URI="http://www.w3.org/2001/04/xmlenc#3des-cbc" />
017       </wsse:Integrity>
018     </wsp:All>
019   </wsp:ExactlyOne>
020 </wsp:Policy>
```

The <wsp:Policy> element (lines 1 and 20) is the top level container element for a policy expression.

The primary operator in the example is <wsp:ExactlyOne> (lines 2, and 19) indicating that the policy consists of two policy statements (two groups of assertions) with the <wsp:ExactlyOne> operator designating that a choice must be made between different sets of assertions.

The subordinate operator in the example is <wsp:All> (lines 3, 10, 11, and 18), which indicates that the set of child assertions are combined to form a policy statement. These indicate that, in this case two, different statements are available. The <wsp:All> element indicates that all of the contained assertions must be met.

The leaf elements in the expression are assertions (lines 4 to 6, 7 to 9, 12 to 14, and 15 to 17) indicating the specific requirements to be met.

Policy expressions may be identified in several ways. As with WSDL [WSDL] and XML Schema [XMLSchema1], policy expressions can be named using URI or QNAME mechanisms. It should be noted that compliance does not require support for both URIs and QNames. That is, it is compliant to specify policies using URIs only, QNames only, or to support both.

A policy expression may be assigned a QName using the *Name* attribute, which specifies the local part of the policy expression's QName. The namespace component of the policy expression's QName may be indicated either using the *TargetNamespace* attribute of the <wsp:Policy> element or may be inherited from a containing element (e.g., <wsdl:definitions>, <xsd:schema>). If no target namespace is specified by either the container or the <wsp:Policy> element, the namespace part of the expression's QName is "".

A policy expression can also itself be a Web resource, hence identifiable by a URI. To allow policy expressions to be embedded in arbitrary containing elements, the *wsu:Id* attribute may be used to indicate a fragment ID.

The next section defines the syntax for policy assertions and the following section defines the policy element and the operators.

3.2 Policy Assertions

A *policy assertion* conveys a requirement, preference, or capability of a given policy subject. Assertions are typed and can be simple or complex. A simple assertion is defined to be an assertion that can be compared to another assertion of the same type based on value-equivalence with no special consideration of the assertion's semantics.

Complex assertions are defined as assertions that require an assertion type-specific means of comparison. The mechanisms for describing these complex type-specific comparisons are specific to the domain-specific policy language in which they are expressed and are outside the scope of this document.

An assertion type MAY be defined such that the assertion is parameterized. For example, an assertion describing the maximum acceptable message size (in bytes) would likely accept an integer parameter indicating the maximum byte count. In contrast, an assertion that simply indicates that TCP-large-window-extensions are required does not need parameters; its mere presence is enough to convey the assertion.

Interactions, if any, between simple or complex assertions within a domain (i.e., an XML namespace) are specific to that domain. We assume there are no interactions between assertions from different domains.

Note that in the XML serialization of a policy, policy assertions are strongly typed, and their type names are QNames that MUST be consistently interpreted independent of subject.

3.2.1 Assertion Usage

Each assertion is associated with a usage type. The usage stipulates how the assertion should be interpreted in relation to the overall policy. For example, one assertion could provide privacy guarantees for information exchanged by the Web services. Another assertion could be a requirement for encryption. These two assertions differ. The first has no externally visible manifestation and the second does – it indicates a requirement for messages being sent to the service. The first is simply a declaration that a service will do (or in this case not do) something. The second requires cooperation between the two Web services. To capture the nature of these differences, WS-Policy defines five usage qualifiers: Required, Optional, Rejected, Observed and Ignored.

Operators that group together assertions are also allowed to designate a usage type in lieu of specifying usage type on each individual assertion.

3.2.2 Policy Assertion Syntax

A policy specifies one or more policy assertions. These individual declarations are domain-specific (e.g., security, transactions) and are expected to be defined in separate domain-specific specifications. Since assertions are expressed as elements, new assertions can be created using namespace extensibility. An assertion is identified by QName and is processed as part of a policy Infoset independent of a specific serialization.

However, every assertion MUST have an in-scope value for the wsp:Usage attribute to qualify its semantics as applied to the policy target; there is no implied value for this attribute. An assertion can be “wsp:Required”, “wsp:Rejected”, “wsp:Optional”, “wsp:Observed”, or “wsp:Ignored” by a service endpoint. The meaning of these categories is explained below as are scoping rules for the wsp:Usage attribute.

Policies are represented using the <wsp:Policy> element. The schema outline for this element is as follows:

```
<wsp:Policy wsu:Id="..."? Name="..."? TargetNamespace="..."? >
    <Assertion wsp:Usage="..."? wsp:Preference="..."? /> *
    ...
    <wsse:Security>...</wsse:Security> ?
</wsp:Policy>
```

The following describes the attributes and elements listed in the schema outlined above:

/wsp:Policy

This element contains policy statements.

/wsp:Policy/@wsu:Id

This optional attribute uniquely identifies the policy element as a URI. The URI is formed by using the Id as a local name relative to the XML base (i.e., base#Id).

/wsp:Policy/@Name

This optional attribute uniquely identifies the policy element as a QName. The QName is formed by using the Name as a local name composed with a target namespace URI. The target namespace URI can be inherited from the context or specified by the TargetNamespace attribute.

/wsp:Policy/@TargetNamespace

This optional attribute of type xsd:anyURI specifies the namespace URI for the policy expression’s QName in the event that one is not available in the surrounding context, or if a different base is desired. The intent is that when this attribute is present the TargetNamespace/Name pair specifies the QName that is the logical identifier of the policy.

/wsp:Policy/Assertion

This is a place holder element name.

/wsp:Policy/Assertion/*@wsp:Usage*

This optional QName attribute identifies how the policy assertion is processed. The following table describes the different values of this attribute. Additionally, the attribute is extensible, additional QName values in addition to those listed may be used.

Value	**Meaning**
wsp:Required	The assertion must be applied to the subject. If the subject does not meet the criteria expressed in the assertion a fault or error will occur.
wsp:Rejected	The assertion is explicitly not supported and if present will cause

	failure.
wsp:Optional	The assertion may be made of the subject but it is not required to be applied.
wsp:Observed	The assertion will be applied to all subjects and requesters of the service are informed that the policy will be applied.
wsp:Ignored	The assertion is processed, but ignored. That is, it can be specified, but no action will be taken as a result of it being specified. Subjects and requesters are informed that the policy will be ignored.

/wsp:Policy/Assertion/*@wsp:Preference*

This optional attribute specifies the preference of this particular alternative. The preference is expressed as an xsd:int. The higher the value of the preference, the greater the weighting of the expressed preference. If no preference is specified, a value of zero is assumed.

/wsp:Policy/wsse:Security

This optional element allows security information to be specified about this policy. This element is defined in WS-Security [WS-Security].

/wsp:Policy/{any}

Additional child elements MAY be specified to make additional policy assertions but MUST NOT contradict the semantics of the parent element; if an element is not recognized, it SHOULD be ignored.

/wsp:Policy/@{any}

Additional attributes MAY be specified but MUST NOT contradict the semantics of the owner element; if an attribute is not recognized, it SHOULD be ignored.

It is legal for a policy expression to contain only assertions with no operators. When no operators are specified, it is implied that there are no alternatives. The following example illustrates a policy expression containing only assertions.

It is not legal for a policy assertion to contain an operator, since operators are used to express relationships between assertions.

```
<wsp:Policy xmlns:wsse="...">
  <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="1">
    <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
  </wsse:SecurityToken>
  <wsse:Integrity wsp:Usage="wsp:Required" wsp:Preference="8" >
    <wsse:Algorithm Type="wsse:AlgSignature"
        URI="http://www.w3.org/2000/09/xmlenc#aes" />
  </wsse:Integrity>
</wsp:Policy>
```

The following example illustrates how to associate a policy expression with a URI:

```
<wsp:Policy xmlns:wsse="..." wsu:Id="P1" xmlns:wsu="...">
```

```
  <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="1" >
    <wsse:TokenType> wsse:Kerberosv5TGT</wsse:TokenType>
  </wsse:SecurityToken>
  <wsse:Integrity wsp:Usage="wsp:Required" wsp:Preference="8" >
    <wsse:Algorithm Type="wsse:AlgSignature"
                    URI="http://www.w3.org/2000/09/xmlenc#aes" />
  </wsse:Integrity>
</wsp:Policy>
```

In the above example, if the XML base was “http://fabrikam123.com/policies”, then the URI for the policy would be “http://fabrikam123.com/policies#P1”.

Alternatively, a policy expression can be identified using QNames:

```
<wsp:Policy xmlns:wsse="..."
            Name="Q1"
            TargetNamespace="http://fabrikam123.com/policies">
  <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="1">
    <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
  </wsse:SecurityToken>
  <wsse:Integrity wsp:Usage="wsp:Required" wsp:Preference="8">
    <wsse:Algorithm Type="wsse:AlgSignature"
                    URI="http://www.w3.org/2000/09/xmlenc#aes" />
  </wsse:Integrity>
</wsp:Policy>
```

In the above example, the QName for the policy would be <f123:Q1> where f123 has the value “http://fabrikam123.com/policies”.

3.3 Policy Operators

In order to author more flexible policies, policy operators are provided which allow alternative acceptable policy statements to be authored. The operators are:

- <wsp:All> which requires that all of its child elements be satisfied
- <wsp:ExactlyOne> which requires that exactly one of its child elements be satisfied
- <wsp:OneOrMore> which requires that at least one of its child elements be satisfied

<wsp:Policy> whose semantics are the same as <wsp:All>

The schema outline for the <wsp:Policy> element is as follows:

```
<wsp:Policy>
  ...
  <wsp:ExactlyOne wsp:Usage="..."? wsp:Preference="..."?>
```

```
    ...
  </wsp:ExactlyOne>
  <wsp:All wsp:Usage="..."? wsp:Preference="..."?> ... </wsp:All>
  <wsp:OneOrMore wsp:Usage="..."? wsp:Preference="..."? >
    ...
  </wsp:OneOrMore>
  ...
</wsp:Policy>
```

The following describes the attributes and elements listed in the schema outlined above:

/wsp:Policy

Policy is top level container for the set of policy operators and assertions.

/wsp:Policy/.../wsp:ExactlyOne

This element may contain one or more policy assertions, references, or operators. It requires that exactly one of its children be satisfied.

/wsp:Policy/.../wsp:All

This element may contain one or more policy assertions, references, or operators. It requires that every one of its children be satisfied.

/wsp:Policy/.../wsp:OneOrMore

This element may contain one or more policy assertions, references, or operators. It requires that at least one of its children be satisfied.

Each operator allows additional child and/or attribute content. Additional child elements and attributes MUST NOT contradict the semantics of the parent / owner element. Additional child elements MUST be assertions, references, or operators. If an attribute is not recognized, it SHOULD be ignored.

In the example below an authentication mechanism must be selected. Either security option “Kerberosv5TGT” or “X509v3” is allowed. The `<wsp:ExactlyOne>` element is provided to allow an exclusive choice between the assertions. The intent is to allow services to indicate a range of supported options for other parties to choose from. As well, the policy indicates that auditing is observed. This is because the `<wsp:Policy>` container has the same behavior as the `<wsp:All>` operator. In this example a preference is not indicated.

```
<wsp:Policy xmlns:wssx="...">
  <wsp:ExactlyOne>
    <wsse:SecurityToken wsp:Usage="wsp:Required">
      <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Usage="wsp:Required">
      <wsse:TokenType>wsse:X509v3</wsse:TokenType>
    </wsse:SecurityToken>
```

```
  </wsp:ExactlyOne>
  <wssx:Audit wsp:Usage="wsp:Observed"/>
</wsp:Policy>
```

In the next example below an authentication mechanism must be selected but the provider indicates which mechanism it “prefers” by indicating a preference. Either security token type “Kerberos5TGT” or “X509v3” is allowed.

```
<wsp:Policy xmlns:wsse="...">
  <wsp:ExactlyOne>
    <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="100">
      <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Usage="wsp:Required" wsp:Preference="1">
      <wsse:TokenType>wsse:X509v3</wsse:TokenType>
    </wsse:SecurityToken>
  </wsp:ExactlyOne>
</wsp:Policy>
```

The <wsp:ExactlyOne> element exists to produce *N* policy statements to facilitate policy selection.

The following example illustrates the representation of a group of assertions as part of an expression with an <wsp:ExactlyOne> operator. In this example, the security profile consists of domain specific policy assertions. The first alternative specifies Kerberos Authentication and Privacy. The second alternative specifies password authentication and an audit trail. Either the first or second alternative must be chosen, but not both. The preference attribute establishes a preference for the first alternative.

```
<wsp:Policy xmlns:wssx="...">
  <wsp:ExactlyOne>
    <wsp:All wsp:Usage="wsp:Required" wsp:Preference="100">
      <wsse:SecurityToken>
        <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
      </wsse:SecurityToken>
      <wssx:Privacy />
    </wsp:All>
    <wsp:All wsp:Preference="1" wsp:Usage="wsp:Required">
      <wsse:SecurityToken>
        <wsse:TokenType>wsse:UsernameToken </wsse:TokenType>
      </wsse:SecurityToken>
```

```
      <wssx:Audit />
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Note that in this example, the *wsp:Usage* and *wsp:Preference* attributes may appear on a policy operator instead of on the assertions themselves. The wsp:Usage attribute applies to the element upon which it appears and to all of that element's descendants, unless overridden by another wsp:Usage attribute on a descendant element.

3.4 Forming Policy from Policy Statements

A policy is the collection of all acceptable policy statements for a specific domain. This collection can be viewed as a table, in which each column is an assertion and each row is a policy statement. For simple assertion types (and single collection), the cells in the table indicate the Boolean predicate of the assertion as defined the previous section.

For example, the policy "either only an x509 certificate and 3DES encryption or a Kerberos certificate and AES are acceptable and all other combinations are invalid" can be represented in a table as:

	Assertions			
Acceptable Policy Statements	**X509**	**Kerberos**	**3DES**	**AES**
Stmt 1	True	False	True	False
Stmt 2	False	True	False	True

This table has two rows. The first row states "x509 is required and 3DES is required and that other aspects are invalid." The second row states "Kerberos is required and AES is required and that other aspects are invalid." Each row is an acceptable alternative, so this table can be read as the intended policy statement above: "either an x509 certificate and 3DES encryption (only) or a Kerberos certificate and AES are acceptable (only)."

3.5 Simple Assertions and Usage Types

A policy statement combines one or more policy assertions to describe an acceptable combination of assertions with their respective usages.

For example, consider the case in which two fictitious assertion types have been defined: X509 and 3DES. For the purposes of this example, we will assume that each assertion has no additional parameters.

A simple policy statement consisting of both assertions would look like this:

Assertion Type	**X509**	**3DES**
Usage	Required	Required

This statement indicates that both X509 and 3DES are required by the (yet to be specified) subject of this statement.

It is also possible to vary the usage for each assertion. For example, the following policy statement indicates that X509 is required by the subject, but that 3DES is explicitly rejected:

Assertion Type	**X509**	**3DES**
Usage	Required	Rejected

To simplify the analysis and processing of policy statements, non-parameterized simple assertions can be viewed as Boolean predicates, with a Required usage resulting in true and Rejected resulting in false.

For example, the second example just shown could be described like this:

Simplified Statements	**X509**	**3DES**
Stmt 1	True	False

It should be noted that complex and parameterized assertion types require more sophisticated logic and that must be handled in a type-specific manner – the specifics of which are outside the scope of this document.

Using this simplified Boolean formulation, a simple assertion with Optional usage results in two statements being induced: one with a true predicate and the other with a false predicate. For example, had the X509 assertion been marked optional and the 3DES assertion marked required, the resultant policy statements would be this:

Assertion Type	**X509**	**3DES**
Stmt 1	True	True
Stmt 2	False	True

Note that either of these two policy statements would be acceptable to the subject of the two assertions just described.

A simple policy statement consisting of fictitious assertions for Privacy and Audit might look like this:

Assertion Type	**Privacy**	**Audit**
Usage	Observed	Required

Following this simplified Boolean formulation, a simple assertion with Observed usage always resolves to true. For example, had there been a privacy assertion marked Observed and an Audit assertion marked Required, the resultant policy statements would be this:

Simple Statements	**Privacy**	**Audit**
Stmt 1	True	True

Continuing the formulation, a simple assertion with Rejects usage always resolves to false would be represent as this:

Assertion Type	**Privacy**	**Audit**
Usage	Observed	Rejects

For example, had there been a privacy assertion marked Rejects the resultant policy statements would be this:

Simple Statements	**Privacy**	**Audit**
Stmt 1	True	False

4. Policy Inclusion

In order to share assertions across policy expressions, the <wsp:PolicyReference> element can be specified as a policy assertion. This element is used to include the content of one policy expression in another expression.

When a <wsp:PolicyReference> element references another <wsp:Policy> element, then the semantics of inclusion are simply to replace the <wsp:PolicyReference> element with a <wsp:All> element containing the [children] property of the referenced <wsp:Policy> element. That is, the contents of the referenced policy *conceptually replace* the <wsp:PolicyReference> element and are wrapped in an <wsp:All> operator. (Note: References that have a digest attribute SHOULD be validated before being included). It is legal for a policy operator to contain a <wsp:PolicyReference> (see example below). It is not legal for an assertion element to contain a <wsp:PolicyReference>. If a policy expression needs to reference other policy expressions the valid way is to encapsulate the assertion within an operator and use references within the operator which will be expanded into the policy expression referenced.

The schema outline for the <wsp:PolicyReference> element is as follows:

```
<wsp:Policy>
  ...
  <wsp:PolicyReference URI="..." ?
                       Ref="..." ?
                       Digest="..." ?
                       DigestAlgorithm="..." ? />
  ...
</wsp:Policy>
```

The following describes the attributes and elements listed in the schema outlined above:

/wsp:Policy/.../wsp:PolicyReference

This element references the policy expression that is being applied to all in-scope resources.

/wsp:Policy/.../wsp:PolicyReference/@URI

This optional attribute references a policy using its URI. Either the URI attribute or the Ref attribute MUST be specified.

/wsp:Policy/.../wsp:PolicyReference/@Ref

This optional attribute references a policy using its QName. Either the URI attribute or the Ref attribute MUST be specified.

/wsp:Policy/.../wsp:PolicyReference/@Digest

This optional attribute (of type xsd:base64Binary) specifies the digest of the included policy. This is used to ensure the included policy is the expected policy.

/wsp:Policy/.../wsp:PolicyReference/@DigestAlgorithm

This optional QName attribute specifies the digest algorithms being used. This specification predefines the algorithm below although additional algorithms can be expressed.

QName	**Description**
wsse:Sha1Exc (implied)	The digest is a SHA1 hash over the octet stream resulting from using the Exclusive XML canonicalization defined for XML Signature [XMLSignature].

In the example below two policies include and extend a common policy. In the first example there is a single policy document containing three policy elements. The first element is given an id but not a fully qualified location. The second and third elements reference the first element by using a URI indicating the referenced element is within the document.

```
<wsp:Policy wsu:Id="audit" xmlns:wsu="..." xmlns:wssx="...">
  <wssx:Audit wsp:Usage="wsp:Observed" />
</wsp:Policy>

<wsp:Policy xmlns:wsse="...">
  <wsp:PolicyReference URI="#audit" />
  <wsse:SecurityToken wsp:Usage="wsp:Required">
    <wsse:TokenType>wsse:X509v3</wsse:TokenType>
  </wsse:SecurityToken>
</wsp:Policy>

<wsp:Policy xmlns:wsse="...">
  <wsp:PolicyReference URI="#audit" />
  <wsse:SecurityToken wsp:Usage="wsp:Required">
    <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
  </wsse:SecurityToken>
</wsp:Policy>
```

The following example illustrates the same thing using QName references:

```
<wsp:Policy Name="audit"
            TargetNamespace="http://fabrikam123.com/policies"
            xmlns:wssx="...">
  <wssx:Audit wsp:Usage="wsp:Observed" />
</wsp:Policy>

<wsp:Policy xmlns:wsse="..."
            xmlns:f123="http://fabrikam123.com/policies">
  <wsp:PolicyReference Ref="f123:audit" />
  <wsse:SecurityToken wsp:Usage="wsp:Required">
    <wsse:TokenType>wsse:X509v3</wsse:TokenType>
  </wsse:SecurityToken>
</wsp:Policy>

<wsp:Policy xmlns:wsse="...">
  <wsp:PolicyReference Ref="f123:audit" />
  <wsse:SecurityToken wsp:Usage="wsp:Required">
    <wsse:TokenType>wsse:Kerberosv5TGT </wsse:TokenType>
  </wsse:SecurityToken>
</wsp:Policy>
```

There are times when it is desirable to “re-use” a portion of a policy expression. Generally, this can be accomplished by placing the common assertions in a separate policy expression and referencing it. However, some assertions may contain sub-elements which can be re-used. For example, a choice among a common set of algorithms. In such cases, it is not possible to use a separate policy expression because the sub-elements being re-used are not complete policy expressions. Consequently, we allow the policy identification mechanisms previously described to be used on policy operators. This allows policy expressions to contain references to these sub-elements and re-use them. When a <wsp:PolicyReference> element reference is to a policy operator, then the <wsp:PolicyReference> element is *replaced* with the element information item being referenced. The following example illustrates this usage of inclusion:

```
<wsp:Policy xmlns:wsse="..." xmlns:cus="...">
  <cus:Assert1>
    <wsp:ExactlyOne wsu:Id="opts">
      <cus:Option1 wsp:Usage="wsp:Required" />
      <cus:Option2 wsp:Usage="wsp:Required" />
```

```
      <cus:Option3 wsp:Usage="wsp:Required" />
    </wsp:ExactlyOne>
  </cus:Assert1>

  <cus:Assert2>
    <wsp:PolicyReference URI="#opts" />
  </cus:Assert2>
</wsp:Policy>
```

5. Security Considerations

It is strongly RECOMMENDED that policies and assertions be signed to prevent tampering.

Policies SHOULD NOT be accepted unless they are signed and have an associated security token to specify the signer has the right to “speak for” the scope containing the policy. That is, a relying party shouldn't rely on a policy unless the policy is signed and presented with sufficient credentials to pass the relying parties acceptance criteria.

It should be noted that the mechanisms described in this document could be secured as part of a SOAP message using WS-Security or embedded within other objects using object-specific security mechanisms.

6. Acknowledgements

We would like to thank the following people for their contributions towards this specification:

Erik Christensen, Microsoft
Giovanni Della-Libera, Microsoft
Scott Konersmann, Microsoft
Frank Leymann, IBM
Steve Lucco, Microsoft
Hiroshi Maruyama, IBM
Steve Millet , Microsoft
Henrik Frystyk Nielsen, Microsoft
Keith Stobie, Microsoft
Tony Storey, IBM
Sanjiva Weerawarana, IBM

7. References

[RFC 2119]
“Key words for use in RFCs to Indicate Requirement Levels,” RFC 2119, S. Bradner (editor), March 1997.

[RFC 2396]
“Uniform Resource Identifiers (URI): Generic Syntax,” RFC 2396, T. Berners-Lee, R. Fielding, and L. Masinter (editors), August 1998.

[SOAP11]

"SOAP: Simple Object Access Protocol 1.1," W3C Note, Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, 08 May 2000.

[SOAP12]
"SOAP Version 1.2 Part 1: Messaging Framework," W3C Candidate Recommendation, Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen (editors), 19 December 2002.

[UDDI]
"UDDI Version 3.0," Oasis Published Specification, Tom Bellwood, Luc Clément, David Ehnebuske, Andrew Hately, Maryann Hondo, Yin Leng Husband, Karsten Januszewski, Sam Lee, Barbara McKee, Joel Munter, and Claus von Riegen (editors), 19 July 2002.

[WS-Security]
"Web Services Security (WS-Security)," Bob Atkinson, Giovanni Della-Libera, Satoshi Hada, Maryann Hondo, Phillip Hallam-Baker, Chris Kaler (Editor), Johannes Klein, Brian LaMacchia, Paul Leach, John Manferdelli, Hiroshi Maruyama, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, John Shewchuk, and Dan Simon, 5 April 2002.

[WS-SecurityPolicy]
"Web Services Security Policy Language (WS-SecurityPolicy)", Giovanni Della-Libera, Phillip Hallam-Baker, Maryann Hondo, Tomasz Janczuk, Chris Kaler, Hiroshi Maruyama, Anthony Nadalin (Editor), Nataraj Nagaratnam, Andrew Nash, Rob Philpott, Hemma Prafullchandra, John Shewchuk, Elliot Waingold, and Riaz Zolfonoon, 18 December 2002.

[WSDL]
"Web Services Description Language (WSDL) 1.1," W3C Note, Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, 15 March 2001.

[XML-NS]
"Namespaces in XML," W3C Recommendation, Tim Bray, Dave Hollander, and Andrew Layman (editors), 14 January 1999.

[XMLSchema1]
"XML Schema Part 1: Structures," W3C Recommendation, Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn (editors), 2 May 2001.

[XMLSignature]
"XML-Signature Syntax and Processing," W3C Recommendation, Donald Eastlake, Joseph Reagle, and David Solo (editors), 12 February 2002.

Appendix I. Schema

A normative copy of the XML Schema [XMLSchema1] for WS-Policy constructs may be retrieved by resolving the URI "http://schemas.xmlsoap.org/ws/2002/12/policy". A non-normative copy of that Schema is included in-line below for convenient reference.

```
<xs:schema targetNamespace="http://schemas.xmlsoap.org/ws/2002/12/policy"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
           xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
```

```
          elementFormDefault="qualified"
          blockDefault="#all">
<xs:import namespace="http://schemas.xmlsoap.org/ws/2002/07/utility"
   schemaLocation="http://schemas.xmlsoap.org/ws/2002/07/utility"/>
<xs:attribute name="Usage" type="wsp:OpenUsageType"/>
<xs:simpleType name="OpenUsageType">
  <xs:annotation>
    <xs:documentation>
      Per GXA norms, we allow other values that are not pre-defined.
    </xs:documentation>
  </xs:annotation>
  <xs:union memberTypes="wsp:UsageType xs:QName"/>
</xs:simpleType>
<xs:simpleType name="UsageType">
  <xs:annotation>
    <xs:documentation>
      Pre-defined Usage types that apply to individual assertions.
      Per GXA norms, we use QNames rather than traditional token-based
      enumerations.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:QName">
    <xs:enumeration value="wsp:Required"/>
    <xs:enumeration value="wsp:Optional"/>
    <xs:enumeration value="wsp:Rejected"/>
    <xs:enumeration value="wsp:Ignored"/>
    <xs:enumeration value="wsp:Observed"/>
  </xs:restriction>
</xs:simpleType>
<xs:attribute name="Preference" type="xs:int"/>
<xs:element name="PolicyReference" type="wsp:PolicyReferenceType"/>
<xs:complexType name="PolicyReferenceType">
  <xs:sequence>
    <xs:any namespace="##any"
            processContents="lax"
            minOccurs="0"
```

```
            maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="URI" type="xs:anyURI" use="optional"/>
  <xs:attribute name="Ref" type="xs:QName" use="optional"/>
  <xs:attribute name="Digest" type="xs:base64Binary" use="optional"/>
  <xs:attribute name="DigestAlgorithm" type="xs:QName" use="optional"/>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
<xs:element name="All" type="wsp:Compositor"/>
<xs:element name="OneOrMore" type="wsp:Compositor"/>
<xs:element name="ExactlyOne" type="wsp:Compositor"/>
<xs:complexType name="Compositor">
  <xs:group ref="wsp:CompositorContent" maxOccurs="unbounded"/>
  <xs:attributeGroup ref="wsp:CompositorAndAssertionAttributes"/>
</xs:complexType>
<xs:group name="CompositorContent">
  <xs:choice>
    <xs:element ref="wsp:OneOrMore"/>
    <xs:element ref="wsp:All"/>
    <xs:element ref="wsp:ExactlyOne"/>
    <xs:element ref="wsp:PolicyReference"/>
    <xs:group ref="wsp:PolicyAssertions"/>
    <xs:any namespace="##other" processContents="lax"/>
  </xs:choice>
</xs:group>
<xs:attributeGroup name="CompositorAndAssertionAttributes">
  <xs:attribute ref="wsp:Usage" use="optional"/>
  <xs:attribute ref="wsp:Preference" use="optional"/>
  <xs:attribute ref="wsu:Id" use="optional"/>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:attributeGroup>
<xs:element name="Policy" type="wsp:PolicyExpression"/>
<xs:complexType name="PolicyExpression">
  <xs:group ref="wsp:CompositorContent"
            minOccurs="0"
            maxOccurs="unbounded"/>
```

```
    <xs:attribute name="Name" type="xs:NCName" use="optional"/>
    <xs:attribute name="TargetNamespace" type="xs:anyURI" use="optional"/>
    <xs:attribute ref="wsu:Id" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
</xs:schema>
```