Web Services Policy Framework (WS-Policy)

March 2006

Version 1.2

Authors

Siddharth Bajaj, VeriSign
Don Box, Microsoft
Dave Chappell, Sonic Software
Francisco Curbera, IBM
Glen Daniels, Sonic Software
Phillip Hallam-Baker, VeriSign
Maryann Hondo, IBM
Chris Kaler, Microsoft
Dave Langworthy, Microsoft
Anthony Nadalin, IBM
Nataraj Nagaratnam, IBM
Hemma Prafullchandra, VeriSign
Claus von Riegen, SAP
Daniel Roth, Microsoft
Jeffrey Schlimmer (Editor), Microsoft
Chris Sharp, IBM
John Shewchuk, Microsoft
Asir Vedamuthu, Microsoft
Ümit Yalçınalp, SAP
David Orchard, BEA Systems

Copyright Notice

Abstract

The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe 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 and capabilities.

Composable Architecture

The Web service specifications (WS*) are designed to be composed with each other to provide a rich set of tools for secure, reliable, and/or transacted Web services. 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 a public draft release and is provided for review and evaluation only. The Authors hope to solicit your contributions and suggestions in the near future. The Authors 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.

WS-Policy defines a policy to be a collection of policy alternatives, where each policy alternative is a collection of policy assertions. Some policy assertions specify traditional requirements and capabilities that will ultimately manifest on the wire (e.g., authentication scheme, transport protocol selection). Other policy assertions 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 does not specify how policies 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. WS-PolicyAttachment [WS-PolicyAttachment] defines such mechanisms, especially for associating policy with arbitrary XML elements, WSDL artifacts, and UDDI elements. 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 Infoset called a *policy expression* that contains domain-specific, Web Service policy information.
- A core set of constructs to indicate how choices and/or combinations of domain-specific policy assertions apply in a Web services environment.

WS-Policy is designed to work with the general Web services framework, including WSDL service descriptions [WSDL 1.1] and UDDI service registrations [UDDI API 2.0, UDDI Data Structure 2.0, UDDI 3.0].

1.2 Example

The following example illustrates a policy:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <wsp:ExactlyOne>
(03)        <sp:Basic256Rsa15 />
(04)        <sp:TripleDesRsa15 />
(05)  </wsp:ExactlyOne>
(06) </wsp:Policy>
```

This example illustrates a security policy using assertions defined in WS-SecurityPolicy [WS-SecurityPolicy]. Lines (01-06) represent a policy for the algorithm suite required for performing cryptographic operations with symmetric or asymmetric key-based security tokens.

Lines (02-05) illustrate the Exactly One policy operator. Policy operators group policy assertions into policy alternatives. A valid interpretation of the policy above would be that an invocation of a Web service uses one of the algorithm suite assertions (Lines 03-04) specified.

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].

This specification uses the following syntax within normative outlines:

- The syntax appears as an XML instance, but values in *italics* indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "(" and ")" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- The characters "[" and "]" are used to call out references and property names.
- XML namespace prefixes (see below) are used to indicate the namespace of the element or attribute being defined.

Normative text within this specification takes precedence over normative outlines, which in turn take precedence over the XML Schema [XML Schema] descriptions.

2.2 Extensibility

Within normative outlines, ellipses (i.e., "...") indicate a point of extensibility that allows other Element or Attribute Information Items. Information Items MAY be added at the indicated extension points but MUST NOT contradict the semantics of the Element Information Item indicated by the **[parent]** or **[owner]** property of the extension. If a processor does not recognize an Attribute Information Item, the processor SHOULD ignore it; if a processor does not recognize an Element Information Item, the processor SHOULD treat it as an assertion.

2.3 XML Namespaces

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

```
http://schemas.xmlsoap.org/ws/2004/09/policy
```

A normative copy of the XML Schema [XML Schema Part 1] for WS-Policy constructs may be retrieved by resolving this URI: http://schemas.xmlsoap.org/ws/2004/09/policy/ws-policy.xsd

Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 1: Prefixes and XML Namespaces used in this specification.

Prefix	Namespace	Specification
sp	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	[WS-SecurityPolicy]
wsdl	http://schemas.xmlsoap.org/wsdl/	[WSDL 1.1]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[WS-Security 2004]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	This specification
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[WS-Security 2004]
xs	http://www.w3.org/2001/XMLSchema	[XML Schema Part 1]

2.4 Terminology

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

Policy – A *policy* is a collection of policy alternatives.

Policy Alternative – A *policy alternative* is a collection of policy assertions.

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

Policy Assertion Type – A *policy assertion type* represents a class of policy assertions and implies a schema for the assertion and assertion-specific semantics.

Policy Assertion Parameter – A *policy assertion parameter* qualifies the behavior indicated by a policy assertion.

Policy Vocabulary – The *policy vocabulary* of a policy is the set of all policy assertion types used in the policy.

Policy Expression – A *policy expression* is an XML Infoset representation of a policy, either in a normal form or in an equivalent compact form.

Policy Subject – A *policy subject* is an entity (e.g., an endpoint, message, resource, interaction) with which a policy can be associated.

Policy Scope – A *policy scope* is a collection of policy subjects to which a policy may apply.

Policy Attachment – A *policy attachment* is a mechanism for associating policy with one or more policy scopes.

3. Policy Model

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

This abstract model is independent of how it is represented as an XML Infoset.

3.1 Policy Assertion

A policy assertion identifies a behavior that is a requirement (or capability) of a policy subject. Assertions indicate domain-specific (e.g., security, transactions) semantics and are expected to be defined in separate, domain-specific specifications.

Assertions are strongly typed by the domain authors that define them. The type is identified only by the XML Infoset **[namespace name]** and **[local name]** properties (that is, the qualified name or QName) of the root Element Information Item representing the assertion. Assertions of a given type MUST be consistently interpreted independent of their policy subjects.

Domain authors MAY define that an assertion contains a policy expression as one of its **[children]**. Policy expression nesting is used by domain authors to further qualify one or more specific aspects of the original assertion. For example, security policy domain authors may define an assertion describing a set of security algorithms to qualify the specific behavior of a security binding assertion.

The XML Infoset of an assertion MAY contain a non-empty **[attributes]** property and/or a non-empty **[children]** property. Such content MAY be used to parameterize the behavior indicated by the assertion. For example, an assertion identifying support for a specific reliable messaging mechanism might include an Attribute Information Item to indicate how long an endpoint will wait before sending an acknowledgement.

Domain authors should be cognizant of the processing requirements when defining complex assertions containing additional assertion content or nested policy expressions. Specifically, domain authors are encouraged to consider when the identity of the root Element Information Item alone is enough to convey the requirement (capability).

3.2 Policy Alternative

A policy alternative is a logical construct which represents a potentially empty collection of policy assertions. An alternative with zero assertions indicates no behaviors. An alternative with one or more assertions indicates behaviors implied by those, and only those assertions.

The vocabulary of a policy alternative is the set of all assertion types within the alternative. The vocabulary of a policy is the set of all assertion types used in all the policy alternatives in the policy. An assertion whose type is part of the policy's vocabulary but is not included in an alternative is explicitly prohibited by the alternative.

Assertions within an alternative are not ordered, and thus aspects such as the order in which behaviors (indicated by assertions) are applied to a subject are beyond the scope of this specification.

A policy alternative MAY contain multiple assertions of the same type. Mechanisms for determining the aggregate behavior indicated by the assertions (and their Post-Schema-Validation Infoset (PSVI) content, if any) are specific to the assertion type and are outside the scope of this document.

3.3 Policy

At the abstract level a policy is a potentially empty collection of policy alternatives. A policy with zero alternatives contains no choices; a policy with one or more alternatives indicates choice in requirements (or capabilities) within the policy.

Alternatives are not ordered, and thus aspects such as preferences between alternatives in a given context are beyond the scope of this specification.

Alternatives within a policy may differ significantly in terms of the behaviors they indicate. Conversely, alternatives within a policy may be very similar. In either case, the value or suitability of an alternative is generally a function of the semantics of assertions within the alternative and is therefore beyond the scope of this specification.

3.4 Web services

Applied in the Web services model, policy is used to convey conditions on an interaction between two Web service endpoints. Satisfying assertions in the policy usually results in behavior that reflects these conditions. Typically, the provider of a Web service exposes a policy to convey conditions under which it provides the service. A requester might use this policy to decide whether or not to use the service. A requester may choose any alternative since each is a valid configuration for interaction with the service, but a requester MUST choose only a single alternative for an interaction with a service since each represents an alternative configuration.

A *policy assertion* is *supported* by a requester if and only if the requester satisfies the requirement (or accommodates the capability) corresponding to the assertion. A *policy alternative* is *supported* by a requester if and only if the requester supports all the assertions in the alternative. And, a *policy* is *supported* by a requester if and only if the requester supports at least one of the alternatives in the policy. Note that although policy alternatives are meant to be mutually exclusive, it cannot be decided in general whether or not more than one alternative can be supported at the same time.

Note that a requester may be able to support a policy even if the requester does not understand the type of each assertion in the vocabulary of the policy; the requester only has to understand the type of each assertion in the vocabulary of a *policy alternative*. This characteristic is crucial to versioning and incremental deployment of new assertions because this allows a provider's policy to include new assertions in new alternatives while allowing requesters to continue to use old alternatives in a backward-compatible manner.

4. Policy Expression

To convey policy in an interoperable form, a policy expression is an XML Infoset representation of a policy. The normal form policy expression is the most straightforward

Infoset; equivalent, alternative Infosets allow compactly expressing a policy through a number of constructs.

4.1 Normal Form Policy Expression

To facilitate interoperability, this specification defines a normal form for policy expressions that is a straightforward XML Infoset representation of a policy, enumerating each of its alternatives that in turn enumerate each of their assertions. The schema outline for the normal form of a policy expression is as follows:

```
<wsp:Policy ... >
  <wsp:ExactlyOne>
    ( <wsp:All> ( <Assertion ...> ... </Assertion> )* </wsp:All> )*
  </wsp:ExactlyOne>
</wsp:Policy>
```

The following describes the Element Information Items defined in the schema outline above:

/wsp:Policy

A policy expression.

/wsp:Policy/wsp:ExactlyOne

A collection of policy alternatives. If there are no Element Information Items in the **[children]** property, there are no admissible policy alternatives, i.e., no behavior is admissible.

/wsp:Policy/wsp:ExactlyOne/wsp:All

A policy alternative; a collection of policy assertions. If there are no Element Information Items in the **[children]** property, this is an admissible policy alternative that is empty, i.e., no behavior is specified.

*/wsp:Policy/wsp:ExactlyOne/wsp:All/**

XML Infoset representation of a policy assertion.

If an assertion in the normal form of a policy expression contains a nested policy expression, the nested policy expression MUST contain at most one policy alternative.

To simplify processing and improve interoperability, the normal form of a policy expression should be used where practical.

For example, the following is the normal form of the policy expression example introduced earlier.

```
<wsp:Policy
  xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
  <wsp:ExactlyOne>
        <wsp:All>
              <sp:Basic256Rsa15 />
        </wsp:All>
        <wsp:All>
              <sp:TripleDesRsa15 />
```

```
(08)       </wsp:All>
(09)  </wsp:ExactlyOne>
(10) </wsp:Policy>
```

Lines (02-05) and Lines (06-08) express the two alternatives in the policy. If the first alternative is selected, only the Basic 256 RSA 15 algorithm suite [WS-SecurityPolicy] is supported; conversely, if the second alternative is selected, only the 3DES RSA 15 algorithm suite is supported.

4.2 Policy Identification

A policy expression MAY be associated with a URI. The schema outline for attributes to associate a URI is as follows:

```
<wsp:Policy ( Name="xs:anyURI" )?
            ( wsu:Id="xs:ID" )?
            ... >
    ...
</wsp:Policy>
```

The following describes the Attribute Information Items listed and defined in the schema outline above:

/wsp:Policy/@Name

The identity of the policy expression as an absolute URI. If omitted, there is no implied value. This URI MAY be used to refer to a policy from other XML documents using a policy attachment mechanism such as those defined in WS-PolicyAttachment [WS-PolicyAttachment].

/wsp:Policy/@wsu:Id

The identity of the policy expression as an ID within the enclosing XML document. If omitted, there is no implied value. To refer to this policy expression, a URI-reference MAY be formed using this value per Section 4.2 of WS-Security [WS-Security].

The following example illustrates how to associate a policy expression with the absolute URI "http://fabrikam123.com/policies/P1":

```
(01) <wsp:Policy
      Name="http://fabrikam123.com/policies/P1"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)    <!-- Details omitted for readability -->
(03) </wsp:Policy>
```

The following example illustrates how to associate a policy expression with the URI-reference "#P1":

```
(01) <wsp:Policy
      wsu:Id="P1"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
```

```
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" >
(02)  <!-- Details omitted for readability -->
(03) </wsp:Policy>
```

4.3 Compact Policy Expression

To express a policy in a more compact form while still using the XML Infoset, this specification defines three constructs: an attribute to decorate an assertion, semantics for recursively nested policy operators, and a policy reference/inclusion mechanism. Each is described in the subsections below.

To interpret a compact policy expression in an interoperable form, a compact expression may be converted to the corresponding normal form expression by the following procedure:

1. Start with the **[document element]** property D of the Document Information Item of the policy expression. The **[namespace name]** of D is always "http://schemas.xmlsoap.org/ws/2004/09/policy". In the base case, the **[local name]** property of D is "Policy"; in the recursive case, the **[local name]** property of D is "Policy", "ExactlyOne", or "All".
2. Expand Element Information Items in the **[children]** property of D that are policy references per Section 4.3.4 Policy Inclusion.
3. Convert each Element Information Item C in the **[children]** property of D into normal form.
    3. If the **[namespace name]** property of C is "http://schemas.xmlsoap.org/ws/2004/09/policy" and the **[local name]** property of C is "Policy", "ExactlyOne", or "All", C is an expression of a policy operator; normalize C by recursively applying this procedure.
    4. Otherwise the Element Information Item C is an assertion; normalize C per Sections 4.3.1 Optional Policy Assertions and 4.3.2 Policy Assertion Nesting.
4. Apply the policy operator indicated by D to the normalized Element Information Items in its **[children]** property and construct a normal form per Section 4.3.3 Policy Operators.

Note that an implementation may use a more efficient procedure and is not required to explicitly convert a compact expression into the normal form as long as the processing results are indistinguishable from doing so.

4.3.1 Optional Policy Assertions

To indicate that a policy assertion is optional, this specification defines an attribute that is a syntactic shortcut for expressing policy alternatives with and without the assertion. The schema outline for this attribute is as follows:

```
<Assertion ( wsp:Optional="xs:boolean" )? ...> ... </Assertion>
```

The following describes the Attribute Information Item defined in the schema outline above:

/Assertion/@wsp:Optional

If true, the expression of the assertion is semantically equivalent to the following:

```
<wsp:ExactlyOne>
```

```
  <wsp:All> <Assertion ...> ... </Assertion> </wsp:All>
  <wsp:All />
</wsp:ExactlyOne>
```

If false, the expression of the assertion is semantically equivalent to the following:

```
<wsp:ExactlyOne>
  <wsp:All> <Assertion ...> ... </Assertion> </wsp:All>
</wsp:ExactlyOne>
```

Omitting this attribute is semantically equivalent to including it with a value of false. Policy expressions should not include this attribute with a value of false, but policy parsers must accept this attribute with a value of false.

For example, the following compact policy expression:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <sp:IncludeTimestamp wsp:Optional="true" />
(03) </wsp:Policy>
```

is equivalent to the following normal form policy expression:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <wsp:ExactlyOne>
(03)        <wsp:All>
(04)              <sp:IncludeTimestamp />
(05)        </wsp:All>
(06)        <wsp:All />
(07)  </wsp:ExactlyOne>
(08) </wsp:Policy>
```

The `@wsp:Optional` attribute in Line (02) of the first policy expression indicates that the assertion in Line (02) is to be included in a policy alternative whilst excluded from another; it is included in Lines (03-05) and excluded in Line (06). Note that `@wsp:Optional` does not appear in the normal form of a policy expression.

4.3.2 Policy Assertion Nesting

Any policy assertion MAY contain a nested policy expression. The schema outline for a nested policy expression is:

```
<Assertion ...>
  ...
  ( <wsp:Policy ...> ... </wsp:Policy> )?
  ...
```

```
</Assertion>
```

The following describes additional processing constraints on the outline listed above:

/Assertion/wsp:Policy

> This indicates that the assertion contains a nested policy expression. If there is no `wsp:Policy` Element Information Item in the **[children]** property, the assertion has no nested policy expression.

Note: if the schema outline for an assertion type requires a nested policy expression but the assertion does not further qualify one or more aspects of the behavior indicated by the assertion type (i.e., no assertions are needed in the nested policy expression), the assertion MUST include an empty `<wsp:Policy/>` Element Information Item in its **[children]** property; as explained in Section 4.3.3 Policy Operators, this is equivalent to a nested policy expression with a single alternative that has zero assertions. If this is not done then two assertions of the same type will not be compatible and intersection may fail (see Section 4.4 Policy Intersection).

/Assertion//wsp:Policy*

> This specification does not define processing for arbitrary `wsp:Policy` Element Information Items in the descendants of an assertion, e.g., in the **[children]** property of one of the **[children]** as in `<Lorem><Ipsum><wsp:Policy> ... </wsp:Policy></Ipsum></Lorem>`.

Policy assertions containing a nested policy expression are normalized recursively. The nesting of a policy expression (and a `wsp:Policy` child) is retained in the normal form, but in the normal form, each nested policy expression contains at most one policy alternative. If an assertion A contains a nested policy expression E, and if E contains more than one policy alternative, A is duplicated such that there are as many instances of A as choices in E, and the nested policy expression of a duplicate A contains a single choice. This process is applied recursively to the assertions within those choices and to their nested policy expression, if any. Intuitively, if a compact policy is thought of as a tree whose branches have branches etc, in the normal form, a policy is a stump with straight vines.

For example, consider the following compact nested policy expression:

```
<wsp:Policy
  xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
  <sp:TransportBinding>
        <wsp:Policy>
              <sp:AlgorithmSuite>
                    <wsp:Policy>
                          <wsp:ExactlyOne>
                                <sp:Basic256Rsa15 />
                                <sp:TripleDesRsa15 />
                          </wsp:ExactlyOne>
                    </wsp:Policy>
              </sp:AlgorithmSuite>
```

```
(12)                 <sp:TransportToken>
(13)                       <wsp:Policy>
(14)                             <sp:HttpsToken
RequireClientCertificate="false" />
(15)                       </wsp:Policy>
(16)                 </sp:TransportToken>
                     <!-- Details omitted for readability -->
(17)           </wsp:Policy>
(18)  </sp:TransportBinding>
(19) </wsp:Policy>
```

Lines (02-18) in this policy expression contain a single transport binding security policy assertion; within its nested policy expression (Lines 03-17), is an algorithm suite assertion (Lines 04-11) whose nested policy expression (Lines 05-10) contains two policy alternatives (Lines 07-08). Generally, a nested policy expression implies recursive processing; in the example above, the behavior indicated by the transport binding assertion requires the behavior indicated by one of the assertions within the algorithm suite assertion.

The normalized form of this policy is equivalent to the following:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <wsp:ExactlyOne>
(03)        <wsp:All>
(04)              <sp:TransportBinding>
(05)                    <wsp:Policy>
(06)                          <sp:AlgorithmSuite>
(07)                                <wsp:Policy>
(08)                                      <sp:Basic256Rsa15 />
(09)                                </wsp:Policy>
(10)                          </sp:AlgorithmSuite>
(11)                          <sp:TransportToken>
(12)                                <wsp:Policy>
(13)                                      <sp:HttpsToken
RequireClientCertificate="false" />
(14)                                </wsp:Policy>
(15)                          </sp:TransportToken>
                              <!-- Details omitted for readability -->
(16)                    </wsp:Policy>
```

```
(17)                </sp:TransportBinding>
(18)          </wsp:All>
(19)          <wsp:All>
(20)                <sp:TransportBinding>
(21)                      <wsp:Policy>
(22)                            <sp:AlgorithmSuite>
(23)                                  <wsp:Policy>
(24)                                        <sp:TripleDesRsa15 />
(25)                                  </wsp:Policy>
(26)                            </sp:AlgorithmSuite>
(27)                            <sp:TransportToken>
(28)                                  <wsp:Policy>
(29)                                        <sp:HttpsToken
RequireClientCertificate="false" />
(30)                                  </wsp:Policy>
(31)                            </sp:TransportToken>
                                <!-- Details omitted for readability -->
(32)                      </wsp:Policy>
(33)                </sp:TransportBinding>
(34)          </wsp:All>
(35)    </wsp:ExactlyOne>
(36) </wsp:Policy>
```

In the listing above, the transport binding and its nested policy expression have been duplicated once for each of the nested alternatives in Lines (07-08) of the compact policy. The first alternative (Lines 03-18) contains a single nested algorithm suite alternative (Line 08) as does the second alternative (Lines 19-34 and 24).

4.3.3 Policy Operators

To compactly express complex policies, policy operators MAY be recursively nested; that is, one or more instances of `wsp:Policy`, `wsp:All`, and/or `wsp:ExactlyOne` MAY be nested within `wsp:Policy`, `wsp:All`, and/or `wsp:ExactlyOne`.

The following rules are used to transform a compact policy expression into a normal form policy expression:

Equivalence

`wsp:Policy` is equivalent to `wsp:All`.

Empty

- `<wsp:All />` expresses a policy with zero policy assertions. Note that since `wsp:Policy` is equivalent to `wsp:All`, `<wsp:Policy />` is therefore equivalent to `<wsp:All />`, i.e., a policy alternative with zero assertions.
- `<wsp:ExactlyOne />` expresses a policy with zero policy alternatives.

Commutative

In line with the previous statements that policy assertions within a policy alternative and policy alternatives within a policy are not ordered (see 3.2 Policy Alternative and 3.3 Policy, respectively), `wsp:All` and `wsp:ExactlyOne` are commutative. For example,

```
<wsp:All> <!-- assertion 1 --> <!-- assertion 2 --> </wsp:All>
```

is equivalent to:

```
<wsp:All> <!-- assertion 2 --> <!-- assertion 1 --> </wsp:All>
```

and:

```
<wsp:ExactlyOne>
  <!-- assertion 1 --> <!-- assertion 2 -->
</wsp:ExactlyOne>
```

is equivalent to:

```
<wsp:ExactlyOne>
  <!-- assertion 2 --> <!-- assertion 1 -->
</wsp:ExactlyOne>
```

Associative

`wsp:All` and `wsp:ExactlyOne` are associative. For example,

```
<wsp:All>
  <!-- assertion 1 -->
  <wsp:All> <!-- assertion 2 --> </wsp:All>
</wsp:All>
```

is equivalent to:

```
<wsp:All> <!-- assertion 1 --> <!-- assertion 2 --> </wsp:All>
```

and:

```
<wsp:ExactlyOne>
  <!-- assertion 1 -->
  <wsp:ExactlyOne> <!-- assertion 2 --> </wsp:ExactlyOne>
</wsp:ExactlyOne>
```

is equivalent to:

```
<wsp:ExactlyOne>
  <!-- assertion 1 --> <!-- assertion 2 -->
</wsp:ExactlyOne>
```

Idempotent

`wsp:All` and `wsp:ExactlyOne` are idempotent. For example,

```
<wsp:All>
  <wsp:All> <!-- assertion 1 --> <!-- assertion 2 --> </wsp:All>
</wsp:All>
```

is equivalent to:

```
<wsp:All> <!-- assertion 1 --> <!-- assertion 2 --> </wsp:All>
```

and:

```
<wsp:ExactlyOne>
  <wsp:ExactlyOne>
    <!-- assertion 1 --> <!-- assertion 2 -->
  </wsp:ExactlyOne>
</wsp:ExactlyOne>
```

is equivalent to:

```
<wsp:ExactlyOne>
  <!-- assertion 1 --> <!-- assertion 2 -->
</wsp:ExactlyOne>
```

Distributive

`wsp:All` distributes over `wsp:ExactlyOne`. For example,

```
<wsp:All>
  <wsp:ExactlyOne>
    <!-- assertion 1 -->
    <!-- assertion 2 -->
  </wsp:ExactlyOne>
</wsp:All>
```

is equivalent to:

```
<wsp:ExactlyOne>
  <wsp:All>
    <!-- assertion 1 -->
  </wsp:All>
  <wsp:All>
    <!-- assertion 2 -->
  </wsp:All>
</wsp:ExactlyOne>
```

Similarly,

```
<wsp:All>
  <wsp:ExactlyOne>
    <!-- assertion 1 -->
    <!-- assertion 2 -->
  </wsp:ExactlyOne>
  <wsp:ExactlyOne>
    <!-- assertion 3 -->
    <!-- assertion 4 -->
  </wsp:ExactlyOne>
</wsp:All>
```

is equivalent to:

```
<wsp:ExactlyOne>
  <wsp:All><!-- assertion 1 --><!-- assertion 3 --></wsp:All>
  <wsp:All><!-- assertion 1 --><!-- assertion 4 --></wsp:All>
  <wsp:All><!-- assertion 2 --><!-- assertion 3 --></wsp:All>
  <wsp:All><!-- assertion 2 --><!-- assertion 4 --></wsp:All>
</wsp:ExactlyOne>
```

Distributing `wsp:All` over an empty `wsp:ExactlyOne` is equivalent to no alternatives. For example,

```
<wsp:All>
  <wsp:ExactlyOne>
    <!-- assertion 1 -->
    <!-- assertion 2 -->
  </wsp:ExactlyOne>
  <wsp:ExactlyOne />
</wsp:All>
```

is equivalent to:

```
<wsp:ExactlyOne />
```

For example, given the following compact policy expression:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <sp:RequireDerivedKeys wsp:Optional="true" />
(03)  <wsp:ExactlyOne>
(04)        <sp:WssUsernameToken10 />
(05)        <sp:WssUsernameToken11 />
(06)  </wsp:ExactlyOne>
(07) </wsp:Policy>
```

Applying Section 4.3.1 Optional Policy Assertions to `@wsp:Optional` in Line (02), and distributing `wsp:All` over `wsp:ExactlyOne` per Section 4.3.3 Policy Operators for the assertions in Lines (04-05) yields:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
```

```
(02)  <wsp:ExactlyOne>
(03)        <wsp:All> <!-- @wsp:Optional alternative with assertion -->
(04)              <sp:RequireDerivedKeys />
(05)        </wsp:All>
(06)        <wsp:All /> <!-- @wsp:Optional alternative without -->
(07)  </wsp:ExactlyOne>
(08)  <wsp:ExactlyOne>
(09)        <wsp:All>
(10)              <sp:WssUsernameToken10 />
(11)        </wsp:All>
(12)        <wsp:All>
(13)              <sp:WssUsernameToken11 />
(14)        </wsp:All>
(15)  </wsp:ExactlyOne>
(16) </wsp:Policy>
```

Note that the assertion listed in Line (02) in the first listing expands into the two alternatives in Lines (03-06) in the second listing.

Finally, noting that `wsp:Policy` is equivalent to `wsp:All`, and distributing `wsp:All` over `wsp:ExactlyOne` yields the following normal form policy expression:

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <wsp:ExactlyOne>
(03)        <wsp:All>
(04)              <sp:RequireDerivedKeys />
(05)              <sp:WssUsernameToken10 />
(06)        </wsp:All>
(07)        <wsp:All>
(08)              <sp:RequireDerivedKeys />
(09)              <sp:WssUsernameToken11 />
(10)        </wsp:All>
(11)        <wsp:All>
(12)              <sp:WssUsernameToken10 />
(13)        </wsp:All>
(14)        <wsp:All>
(15)              <sp:WssUsernameToken11 />
```

```
(16)        </wsp:All>
(17)  </wsp:ExactlyOne>
(18) </wsp:Policy>
```

Note that the two alternatives listed in Lines (03-06) in the second listing are combined with the two alternatives listed in Lines (09-14) in the second listing to create four alternatives in the normalized policy, Lines (03-06), (07-10), (11-13), and (14-16).

4.3.4 Policy Inclusion

In order to share assertions across policy expressions, the `wsp:PolicyReference` element MAY be present anywhere a policy assertion is allowed inside a policy expression. This element is used to include the content of one policy expression in another policy expression.

When a `wsp:PolicyReference` element references a `wsp:Policy` element, then the semantics of inclusion are simply to replace the `wsp:PolicyReference` element with a `wsp:All` element whose **[children]** property is the same as 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 a `wsp:All` operator. Using the `wsp:PolicyReference` element, a policy expression MUST NOT reference itself either directly or indirectly. (Note: References that have a `@Digest` attribute SHOULD be validated before being included.)

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

```
<wsp:Policy>
  ...
  <wsp:PolicyReference
      URI="xs:anyURI"
    ( Digest="xs:base64Binary" ( DigestAlgorithm="xs:anyURI" )? )?
      ... />
  ...
</wsp:Policy>
```

The following describes the Attribute and Element Information Items defined in the schema outline above:

/wsp:Policy/…/wsp:PolicyReference

This element references a policy expression that is being included.

/wsp:Policy/…/wsp:PolicyReference/@URI

This attribute references a policy expression by URI. For a policy expression within the same XML Document, the reference SHOULD be a URI-reference to a policy expression identified by an ID. For an external policy expression, there is no requirement that the URI be resolvable; retrieval mechanisms are beyond the scope of this specification. After retrieval, there is no requirement to check that the retrieved policy expression is associated (Section 4.2 Policy Identification) with this URI. The URI included in the retrieved policy expression, if any, MAY be different than the URI used to retrieve the policy expression.

/wsp:Policy/…/wsp:PolicyReference/@Digest

This optional attribute specifies the digest of the referenced policy expression. This is used to ensure the included policy is the expected policy. If omitted, there is no implied value.

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

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

URI	**Description**
http://schemas.xmlsoap.org/ws/2004/09/policy/Sha1Exc (implied)	The digest is a SHA1 hash over the octet stream resulting from using the Exclusive XML canonicalization defined for XML Signature [XML-Signature].

In the example below two policies include and extend a common policy. In the first example there is a single policy document containing two policy assertions. The expression is given an identifier but not a fully qualified location. The second and third expressions reference the first expression by URI indicating the referenced expression is within the document.

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
      wsu:Id="Protection" >
(02)  <sp:EncryptSignature wsp:Optional="true" />
(03)  <sp:ProtectTokens wsp:Optional="true" />
(04) </wsp:Policy>
```

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
(02)  <wsp:PolicyReference URI="#Protection" />
(03)  <sp:OnlySignEntireHeadersAndBody />
(04) </wsp:Policy>
```

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
```

```
 <sp:IncludeTimestamp />
 <wsp:PolicyReference URI="#Protection" />
 <sp:OnlySignEntireHeadersAndBody />
</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.

4.4 Policy Intersection

Policy intersection is useful when two or more parties express policy and want to limit the policy alternatives to those that are mutually compatible. For example, when a requester and a provider express requirements on a message exchange, intersection identifies compatible policy alternatives (if any) included in both requester and provider policies. Intersection is a commutative, associative function that takes two policies and returns a policy.

Because the set of behaviors indicated by a policy alternative depends on the domain-specific semantics of the collected assertions, determining whether two policy alternatives are compatible generally involves domain-specific processing. As a first approximation, an algorithm is defined herein that approximates compatibility in a domain-independent manner; specifically, for two policy alternatives to be compatible, they must at least have the same vocabulary (see Section 3.2 Policy Alternative).

- Two policy assertions are compatible if they have the same type and
  - If either assertion contains a nested policy expression, the two assertions are compatible if they both have a nested policy expression and the alternative in the nested policy expression of one is compatible with the alternative in the nested policy expression of the other.

Assertion parameters are not part of the compatibility determination defined herein but may be part of other, domain-specific compatibility processing.

- Two policy alternatives are compatible if each assertion in one is compatible with an assertion in the other, and vice-versa. If two alternatives are compatible, their intersection is an alternative containing all of the assertions in both alternatives.
- Two policies are compatible if an alternative in one is compatible with an alternative in the other. If two policies are compatible, their intersection is the set of the intersections between all pairs of compatible alternatives, choosing one alternative from each policy. If two policies are not compatible, their intersection has no policy alternatives.

As an example of intersection, consider two input policies in normal form:

```
<wsp:Policy
 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
 <!-- Policy P1 -->
 <wsp:ExactlyOne>
       <wsp:All> <!-- Alternative A1 -->
```

```
(04)                <sp:SignedElements>
(05)                      <sp:XPath>/S:Envelope/S:Body</sp:XPath>
(06)                </sp:SignedElements>
(07)                <sp:EncryptedElements>
(08)                      <sp:XPath>/S:Envelope/S:Body</sp:XPath>
(09)                </sp:EncryptedElements>
(10)          </wsp:All>
(11)          <wsp:All> <!-- Alternative A2 -->
(12)                <sp:SignedParts>
(13)                      <sp:Body />
(14)                      <sp:Header
      Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
(15)                </sp:SignedParts>
(16)                <sp:EncryptedParts>
(17)                      <sp:Body />
(18)                </sp:EncryptedParts>
(19)          </wsp:All>
(20)  </wsp:ExactlyOne>
(21) </wsp:Policy>
```

The listing above contains two policy alternatives. The first alternative, (Lines 03-10) contains two policy assertions. One indicates which elements should be signed (Lines 04-06); its type is `sp:SignedElements` (Line 04), and its parameters include an XPath expression for the content to be signed (Line 05). The other assertion (Lines 07-09) has a similar structure: type (Line 07) and parameters (Line 08).

The second alternative (Lines 11-19) also contains two assertions, each with type (Line 12 and Line 16) and parameters (Lines 13-14 and Line 17).

As this example illustrates, compatibility between two policy assertions is based on assertion type and delegates parameter processing to domain-specific processing.

```
(01) <wsp:Policy
      xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
      <!-- Policy P2 -->
(02)  <wsp:ExactlyOne>
(03)          <wsp:All> <!-- Alternative A3 -->
(04)                <sp:SignedParts />
(05)                <sp:EncryptedParts>
(06)                      <sp:Body />
(07)                </sp:EncryptedParts>
```

```
        </wsp:All>
        <wsp:All> <!-- Alternative A4 -->
              <sp:SignedElements>
                    <sp:XPath>/S:Envelope/S:Body</sp:XPath>
              </sp:SignedElements>
        </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Because there is only one alternative (A2) in policy P1 with the same vocabulary – the assertions have the same type – as another alternative (A3) in policy P2, the intersection is a policy with a single alternative that contains all of the assertions in A2 and in A3.

```
<wsp:Policy
  xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
  <!-- Intersection of P1 and P2 -->
  <wsp:ExactlyOne>
        <wsp:All>
              <sp:SignedParts >
                    <sp:Body />
                    <sp:Header
  Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
              </sp:SignedParts>
              <sp:EncryptedParts>
                    <sp:Body />
              </sp:EncryptedParts>
              <sp:SignedParts />
              <sp:EncryptedParts>
                    <sp:Body />
              </sp:EncryptedParts>
        </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Note that there are > 1 assertions of the type `sp:SignedParts`; when the behavior associated with `sp:SignedParts` is invoked, the contents of both assertions are used to indicate the correct behavior. Whether these two assertions are compatible depends on the domain-specific semantics of the `sp:SignedParts` assertion. To leverage intersection, assertion authors are encouraged to factor assertions such that two assertions of the same assertion type are always (or at least typically) compatible.

5. Security Considerations

It is 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 [SOAP 1.1, SOAP 1.2] using WS-Security [WS-Security 2004] 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: Dimitar Angelov (SAP), Martijn de Boer (SAP), Erik Christensen (Microsoft), Giovanni Della-Libera (Microsoft), Martin Gudgin (Microsoft), Yigal Hoffner (IBM), Brian Hulse (IBM), Andrew Jones (IBM), Todd Karakashian (BEA), Scott Konersmann (Microsoft), Frank Leymann (IBM), Steve Lucco (Microsoft), Al Lee (Microsoft), David Levin (Microsoft), Ashok Malhotra (formerly Microsoft), Hiroshi Maruyama (IBM), Steve Millet (Microsoft), Vick Mukherjee (Microsoft), Henrik Frystyk Nielsen (Microsoft), Paul Nolan (IBM), Mark Nottingham, (formerly BEA Systems), Bruce Rich (IBM), Keith Stobie (Microsoft), Tony Storey (IBM), Sanjiva Weerawarana (IBM, currently WSO2), Volker Wiechers (SAP).

7. References

[RFC 2119]

S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

[SOAP 1.1]

D. Box, et al, "Simple Object Access Protocol (SOAP) 1.1," May 2000. (See http://www.w3.org/TR/2000/NOTE-SOAP-20000508.)

[SOAP 1.2]

M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.)

[UDDI API 2.0]

D. Ehnebuske, et al, "UDDI Version 2.04 API," July 2002. (See http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.)

[UDDI Data Structure 2.0]

D. Ehnebuske, et al, "UDDI Version 2.03 Data Structure Reference," July 2002. (See http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm.)

[UDDI 3.0]

T. Bellwood, et al, "UDDI Version 3.0," July 2002. (See http://uddi.org/pubs/uddi_v3.htm.)

[WS-PolicyAttachment]

D. Box, et al, "Web Services Policy Attachment (WS-PolicyAttachment)," March 2006. (See http://schemas.xmlsoap.org/ws/2004/09/policy.)

[WS-Security 2004]
A. Nadalin, et al, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)," March 2004. (See http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.)

[WS-SecurityPolicy]
G. Della-Libera, et al, "Web Services Security Policy Language (WS-SecurityPolicy)," July 2005. (See http://www.oasis-open.org/committees/download.php/16569/.)

[WSDL 1.1]
E. Christensen, et al, "Web Services Description Language (WSDL) 1.1," March 2001. (See http://www.w3.org/TR/wsdl.)

[XML Schema Part 1]
H. Thompson, et al, "XML Schema Part 1: Structures Second Edition," October 2004. (See http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.)

[XML-Signature]
D. Eastlake, et al, "XML-Signature Syntax and Processing," February 2002. (See http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.)