Liberty ID-WSF SOAP Binding Specification

Version: 2.0-errata-v1.0

Editors:
Jeff Hodges, NeuStar, Inc.
John Kemp, Nokia Corporation
Robert Aarts, Hewlett-Packard
Greg Whitehead, Hewlett-Packard
Paul Madsen, NTT

Contributors:
Conor Cahill, America Online, Inc.
Darryl Champagne, IEEE-ISTO
Marc Hadley, Sun Microsystems, Inc.
Jukka Kainulainen, Nokia Corporation
Rob Lockhart, IEEE-ISTO
Jonathan Sergent, Sun Microsystems, Inc.

Abstract:

This specification defines a SOAP binding for the Liberty Identity Web Services Framework (ID-WSF) and the Liberty Identity Services Interface Specifications (ID-SIS). It specifies use of the Web Services Addressing (WS-Addressing) SOAP extension, as well as provider declaration, processing context, consent claims, usage directives and a number of other optional headers.

Filename: liberty-idwsf-soap-binding-2.0-errata-v1.0.pdf

Notice

This document has been prepared by Sponsors of the Liberty Alliance. Permission is hereby granted to use the document solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this document for other uses must contact the Liberty Alliance to determine whether an appropriate license for such use is available.

Implementation of certain elements of this document may require licenses under third party intellectual property rights, including without limitation, patent rights. The Sponsors of and any other contributors to the Specification are not and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights. **This Specification is provided "AS IS", and no participant in the Liberty Alliance makes any warranty of any kind, express or implied, including any implied warranties of merchantability, non-infringement of third party intellectual property rights, and fitness for a particular purpose.** Implementers of this Specification are advised to review the Liberty Alliance Project's website (http://www.projectliberty.org/) for information concerning any Necessary Claims Disclosure Notices that have been received by the Liberty Alliance Management Board.

Liberty Alliance Project
Licensing Administrator
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08855-1331, USA
info@projectliberty.org

Contents

1. Introduction

The Liberty Identity Web Services Framework (ID-WSF) [LibertyIDWSFOverview] is designed so that "application layer" messages or "services" messages utilizing the framework, referred to as *ID-* messages* in this specification, may be mapped onto various transport or transfer protocols. Thus, they are designed to be conveyed in the data portion of the underlying protocol's messages. ID-* messages do not intrinsically address specific aspects of message exchange such as: to which system entity the message is to be sent, message correlation, the mechanics of message exchange, or security context.

Examples of ID-* messages include the `<DiscoveryLookupRequest>` message of [LibertyDisco], and the `<Modify>` message of [LibertyIDPP].

This specification defines a mapping of ID-* messages onto SOAP [SOAPv1.1], an XML-based [XML] messaging protocol.

SOAP itself does not define the specific message exchange aspects mentioned above, but offers an *extensibility model* that may be used to define message components that do address such message exchange specifics. SOAP extensibility is effected by adding message components to the portion of the SOAP message called the *Header*. These message components are referred to as *SOAP header blocks* [SOAPv1.2].

WS-Addressing SOAP Binding [WSAv1.0-SOAP] is a SOAP extension that defines a set of SOAP header blocks that facilitate end-to-end addressing and message correlation. This specification profiles WSAv1.0-SOAP to address specific aspects of ID-* message exchange functionality.

This specification also defines several optional SOAP header blocks relevant to ID-* message processing. They are:

- Processing Context:

  An ID-* requester may need to express additional context for a given request, for example indicating that the requester expects to make such requests in the future when the Principal may or may not be online. This specification defines the `<ProcessingContext>` header block for this purpose.

- Consent Claims:

  ID-WSF-based entities may wish to claim whether they obtained the Principal's consent for carrying out any given operation, such as updating a Principal's Personal Profile entry [LibertyIDPP]. This specification defines the `<Consent>` header block for this purpose.

- Credentials Context:

  The receiver of an ID-* message might indicate that credentials supplied in the request did not meet its policy in allowing access to the requested resource. The `<CredentialsContext>` header block allows such policies to be expressed to the requester.

- Endpoint Update:

  The `<EndpointUpdate>` header block allows a service to indicate that requesters should contact it on a different endpoint or use a different security mechanism and credentials to access the requested resource.

- Timeout:

  The `<Timeout>` header block is defined in this specification to allow the receiver of an ID-* message to indicate that processing of the received message failed due to a timeout condition.

- Usage Directives:

  ID-WSF-based entities may wish to indicate their policies for handling data at the time of data request, and entities releasing data may wish to specify their policies for the subsequent use of data at the time of data release. This specification defines the `<UsageDirective>` header block for this purpose.

- Application EPR:

  This specification defines the `<ApplicationEPR>` header block as a means for a sender to specify application endpoints that may be referenced from the SOAP Body of the message.

- User Interaction:

  A WSC that interacts with a user (typically through a web-site offered by the WSC) may need to indicate its readiness to redirect the user agent of the user, or its readiness to pose questions to the user on behalf of other parties (such as WSPs). This specification defines the `<UserInteraction>` header block for this purpose.

Additionally, this specification defines how ID-* messages are bound into SOAP message bodies, and how the SOAP header blocks implementing the above functionalities are bound into SOAP message headers.

Note that other specifications in the ID-WSF specification suite also define SOAP header blocks, for example [LibertySecMech], which may be used concurrently with the header blocks defined in this specification. Header blocks specified in specifications outside of the ID-WSF specification suite may also be composed with ID-WSF header blocks. An example is the `<wsse:Security>` header block as discussed in [LibertySecMech] . However no further mention of doing such is made in this specification.

2. Notation and Conventions

This specification uses schema documents conforming to W3C XML Schema [Schema1-2] and normative text to describe the syntax and semantics of XML-encoded protocol messages.

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 [RFC2119]:

> " they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions) "

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

2.1. XML Namespaces

This specification makes normative use of the XML namespace prefixes noted in Table 1.

197 **Table 1. XML Namespaces and Prefixes**

Prefix	Namespace
sb:	Represents the Liberty SOAP Binding namespace (v2.0): **`urn:liberty:sb:2006-08`** **Note** This is the point of definition of this namespace. This namespace is the default for instance fragments, type names, and element names in this document when a namespace is not explicitly noted.
sbf:	Represents the Liberty SOAP Binding namespace (cross-version framework): **`urn:liberty:sb`** **Note** This is the point of definition of this namespace.
idpp:	Represents the namespace defined in [LibertyIDPP].
is:	Represents the namespace defined in [LibertyInteract].
S:	Represents the SOAP namespace: `http://schemas.xmlsoap.org/soap/envelope/` This namespace is defined in [SOAPv1.1].
samlp:	Represents the namespace defined in [SAMLCore2].
wsa:	Represents the WS-Addressing namespace: `http://www.w3.org/2005/08/addressing` This namespace is defined in [WSAv1.0].
wsse:	Represents the SOAP Message Security namespace: `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-secext-1.0.xsd` This namespace is defined in [wss-sms].
wsu:	Represents the SOAP Message Security Utility namespace: `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd` This namespace is defined in [wss-sms].
xs:	Represents the W3C XML schema namespace: `http://www.w3.org/2001/XMLSchema` This namespace is defined in [Schema1-2].

198 ## 2.2. Terminology

199 This section defines key terminology used in this specification. Definitions for other Liberty-specific terms can be
200 found in [LibertyGlossary]. See also [RFC2828] for overall definitions of security-related terms.

201 affiliation
202 An *affiliation* is a set of one or more entities, described by *Provider IDs*, who may perform
203 Liberty interactions as a member of the set. An affiliation is referenced by exactly one
204 *Affiliation ID*, and is administered by exactly one entity identified by their Provider ID.
205 Members of an affiliation may invoke services either as a member of the affiliation–by virtue
of their Affiliation ID, or individually by virtue of their Provider ID [LibertyGlossary].

Affiliation ID	An *Affiliation ID* identifies an *affiliation*. It is schematically represented by the `affiliationID` attribute of the `<AffiliationDescriptor>` metadata element [LibertyMetadata].
client	A *role* assumed by a *system entity* which makes a request of another system entity, often termed a *server* [RFC2828], i.e., a client is also a *sender*.
ID-*	A shorthand designator referring to the Liberty ID-WSF, ID-FF, and ID-SIS specification sets. For example, one might say that the former specification sets are all part of the Liberty ID-* specification suite.
ID-* header block	One of the header blocks defined in this specification, or defined in any of the other Liberty ID-* specification suite.
ID-* message	Equivalent to *ordinary ID-* message*.
ID-* fault message	See Section 4.4.
ID-SIS	Liberty Identity Service Interface specification set.
ID-WSF	Liberty Identity Web Services Framework specification set.
MEP	see Message Exchange Pattern.
Message Exchange Pattern	A [SOAPv1.2] term for the overall notion of various patterns of message exchange between SOAP nodes. For example, request-reply and one-way are two *MEPs* used in this specification.
message thread	A *message thread* is an exchange of messages in a request-response *MEP* between two *SOAP nodes*. All the messages of a given message thread are "linked" via each message's `<wsa:RelatesTo>` header block value being set, by the sender, from the previous successfully received message's `<wsa:MessageID>` header block value.
Ordinary ID-* message	See Section 4.3.
processing context	A *processing context* is the collection of specific circumstances under which a particular processing step or set of steps take place.
processing context facet	A *processing context facet* is an identified aspect, inherent or additive, of a *processing context*.
provider	A *provider* is a Liberty-enabled entity that performs one or more of the provider roles in the Liberty architecture, for example Service Provider or Identity Provider. See also *Liberty-enabled Provider* in [LibertyGlossary]. Providers are identified in Liberty protocol interactions by their *Provider IDs* or optionally their *Affiliation ID* if they are a member of an affiliation(s) and are acting in that capacity.
Provider ID	A *Provider ID* identifies an entity known as a *provider*. It is schematically represented by the `providerID` attribute of the `<EntityDescriptor>` metadata element [LibertyMetadata].
receiver	A *role* taken by a *system entity* when it receives a message sent by another system entity. See also *SOAP receiver* in [SOAPv1.2].
role	A function or part performed, especially in a particular operation or process [Merriam-Webster].

sender	A *role* donned by a *system entity* when it constructs and sends a message to another system entity. See also *SOAP sender* in [SOAPv1.2].
server	A *role* performed by a *system entity* that provides a service in response to requests from other system entities called *clients* [RFC2828]. Note that in order to provide a service to clients; a server will often be both a *sender* and a *receiver*.
service request	A *service request* is another term for an *ordinary ID-* message*. Service request is also loosely equivalent to a "SOAP-bound (ordinary) ID-* message."
SOAP-bound ID-* message	See Section 4.5.
SOAP header block	A [SOAPv1.2] term whose definition is: An [element] used to delimit data that logically constitutes a single computational unit within the SOAP header. In [SOAPv1.1] these are known as simply *SOAP headers*, or simply *headers*. This specification uses the SOAPv1.2 terminology.
SOAP message	In this specification, the term *SOAP message* refers to a message consisting of only a `<S:Envelope>` element as defined in [SOAPv1.1]. It contains two top-level subelements: `<S:Header>` and `<S:Body>`. This message is in turn mapped onto a lower-layer transport or transfer protocol, typically HTTP [RFC2616].
SOAP node	A [SOAPv1.2] term describing *system entities* who are parties to SOAP-based message exchanges that are, for purposes of this specification, also the ultimate destination of the exchanged messages, i.e., *SOAP endpoints*. In [SOAPv1.1], SOAP nodes are referred to as *SOAP endpoints*, or simply *endpoints*. This specification uses the SOAPv1.2 terminology.
system entity	An active element of a computer/network system. For example, an automated process or set of processes, a subsystem, a person or group of persons that incorporates a distinct set of functionality [SAMLGloss2].

2.3. Treatment of Boolean Values

For readability, when an XML Schema type is specified to be `xsd:boolean`, this document discusses the values as `TRUE` and `FALSE` rather than `"1"` and `"0"`, which will exist in a document instance conforming to the SOAP Envelope 1.1 schema [SOAPv1.1-Schema].

2.4. String and URI Values

All string and URI [RFC3986] values in this specification have the types `string` (as a `base` type in this case) and `anyURI` respectively, which are built in to the W3C XML Schema Datatypes specification [Schema2-2]. All strings in ID-WSF messages MUST consist of at least one non-whitespace character (whitespace is defined in the XML Recommendation [XML] section 2.3). Empty and whitespace-only values are disallowed. Also, unless otherwise indicated in this specification, all URI values MUST consist of at least one non-whitespace character.

Note

Various element and/or attribute components of the schema described by this specification (see Appendix A: SOAP Binding Schema XSD , below) may have further requirements placed on the values they may take on. For example, see Section 5.2.1: `<wsa:MessageID>` Value Requirements .

2.5. Time Values

All time values in this specification have the type `dateTime`, which is built in to the W3C XML Schema Datatypes specification [Schema2-2] and MUST be expressed in UTC form.

Senders and receivers SHOULD NOT rely on other applications supporting time resolution finer than milliseconds. Implementations MUST NOT generate time instants that specify leap seconds.

3. Schema Particulars

This section addresses schema particulars such as which schemas this specification defines, describes, and depends upon, as well as various underlying schema types.

3.1. Schema Declarations

This specification normatively defines and describes an XML schema which is constituted in the XML Schema [Schema1-2] files ("Liberty ID-WSF SOAP Binding Schema v2.0," reproduced in Appendix A). In addition, the Liberty ID-WSF SOAP Binding Schema file explicitly includes, in the XML Schema sense, the Liberty ID-WSF utility schema file (reproduced in Appendix C).

Also, the Liberty ID-WSF SOAP Binding Schema files explicitly depend upon the SOAP Message Security Utility 1.0 schema [wss-sms] (reproduced in Appendix E) and Web Services Addressing 1.0 schema [WSAv1.0-Schema] (reproduced in Appendix F).

3.2. "ID" Attributes

The XML Schema [Schema1-2] type `xs:ID` is used to declare *ID* attributes on elements, such as SOAP header blocks, that must be referenceable, say by an XML Signature [LibertySecMech]. It should be noted that XML processors, such as XML Signature verifiers, must be aware of the `xs:ID` type of these ID attributes in order resolve references to the elements they identify.

In this specification, as in Web Services Security and Web Services Addressing specifications on which this specification builds, `xs:anyAttribute` is used on all elements that must be capable of carrying an ID attribute. Interoperability profiles such as the ID-WSF SCR [LibertyIDWSF20SCR] may require use of a particular ID attribute such as `xml:id`. In the absence of such profile requirements `wsu:Id` [wss-sms] MUST be used.

3.3. Status Types

The `<Status>` element, of type `StatusType` complex type, is used in this specification to convey status codes and related information. The schema fragment in Figure 1, from the ID-WSF Utility schema (Appendix C), shows both the `<Status>` element and `StatusType` complex type.

```
<xs:complexType name="StatusType">
   <xs:annotation>
      <xs:documentation>
         A type that may be used for status codes.
      </xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="code" type="xs:string" use="required"/>
   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
   <xs:attribute name="comment" type="xs:string" use="optional"/>
</xs:complexType>

<xs:element name="Status" type="StatusType">
   <xs:annotation>
      <xs:documentation>
         A standard Status type
      </xs:documentation>
   </xs:annotation>
</xs:element>
```

335 **Figure 1. `Status` and `StatusType` Schema**

336 3.3.1. Status Codes

337 This section lists, in Table 2, the values defined in this specification for the `code` attribute of the `<Status>` element.
338 Other specifications MAY define additional code attribute values.

339 **Table 2. Status Codes**

Code	Semantics	Suggested Fault Source
InvalidActor	There is an issue with the `actor` attribute on the indicated header block in the indicated message.	S:Client
InvalidMustUnderstand	There is an issue with the `mustUnderstand` attribute on the indicated header block in the indicated message.	S:Client
StaleMsg	The indicated inbound SOAP-bound ID-* message has a timestamp value outside of the receivers allowable time window.	S:Client
DuplicateMsg	The indicated inbound SOAP-bound ID-* message appears to be a duplicate.	S:Client
InvalidRefToMsgID	The indicated inbound SOAP-bound ID-* message appears to incorrectly refer to the preceding message in the message thread.	S:Client
ProviderIDNotValid	The receiver does not consider the claimed Provider ID to be valid.	S:Client
AffiliationIDNotValid	The receiver does not consider the claimed Affiliation ID to be valid.	S:Client
TargetIdentityNotValid	The receiver does not consider the target identity to be valid.	S:Client
FrameworkVersionMismatch	The framework version used in the conveyed ID-* message does not match what was expected by the reciever.	S:Client
IDStarMsgNotUnderstood	There was a problem with understanding/parsing the conveyed ID-* message.	S:Client
ProcCtxURINotUnderstood	The receiver did not understand the processing context facet URI.	S:Server
ProcCtxUnwilling	The receiver is unwilling to apply the sender's stipulated processing context.	S:Server
CannotHonourUsageDirective	The receiver is unable or unwilling to honor the stipulated usage directive.	S:Server
EndpointUpdated	The request cannot be processed at this endpoint. This is typically used in conjunction with the `<EndpointUpdate>` header block to indicate the endpoint to which the request should be re-submitted.	S:Server
InappropriateCredentials	The sender has submitted a request that does not meet the needs of the receiver. The receiver may indicate credentials that are acceptable to them via a `<CredentialsContext>` or `<EndpointUpdate>` header block.	S:Client
ProcessingTimeout	The sender is indicating that processing of the request has failed due to the processing taking longer than the `maxProcessingTime` specified on the request `<Timeout>` header block.	S:Server
InteractionRequired	the recipient has a need to start an interaction in order to satisfy the service request but the interact attribute value was set to `DoNotInteract`.	S:Server
InteractionRequiredForData	the service request could not be satisfied because the WSP would have to interact with the requesting principal in order to obtain (some of) the requested data but the interact attribute value was set to `DoNotInteractForData`.	S:Server
InteractionTimeNotSufficient	the recipient has a need to start an interaction but has reason to believe that more time is needed that allowed for by the value of the `maxInteractTime` attribute.	S:Server
InteractionTimeout	the recipient could not satisfy the service request due to an unfinished interaction.	S:Server

3.4. SOAP Fault Types

The SOAPv1.1 `Fault` and `detail` complex types are used in this specification to convey processing exceptions.

The schema fragment in Figure 2, extracted from [SOAPv1.1-Schema], defines the SOAPv1.1 `Fault` and `detail` complex types, which define the `<S:Fault>` and `<detail>` elements, respectively.

Note

The `<S:Fault>` element is **not** intended to be used as a SOAP header block. Rather, it is designed to be conveyed in the `<S:Body>` of a SOAP message.

```
  <xs:element name="Fault" type="tns:Fault"/>

  <xs:complexType name="Fault" final="extension">
     <xs:annotation>
        <xs:documentation>
           Fault reporting structure
        </xs:documentation>
</xs:annotation>
     <xs:sequence>
        <xs:element name="faultcode" type="xs:QName"/>
        <xs:element name="faultstring" type="xs:string"/>
        <xs:element name="faultactor" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="detail" type="tns:detail" minOccurs="0"/>
     </xs:sequence>
  </xs:complexType>

  <xs:complexType name="detail">
     <xs:sequence>
        <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
     </xs:sequence>
     <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
```

Figure 2. SOAP `Fault` and `detail` Types Schema

4. SOAP Binding

This section defines the notion of *ID-* messages* and the overall, high-level considerations with respect to *binding* them into *SOAP messages* for subsequent conveyance. The detailed processing rules are then given in Section 5.11: Messaging Processing Rules.

4.1. SOAP Version

This specification normatively depends upon SOAP version 1.1, as specified in [SOAPv1.1]. Messages conformant to this specification MUST also be conformant to [SOAPv1.1].

4.2. The SOAPAction HTTP Header

[SOAPv1.1] defines the SOAPAction HTTP header, and requires its usage on HTTP-bound SOAP messages. This header may be used to indicate the "intent" of a SOAP message to the recipient.

Note

The value of the SOAPAction HTTP header SHOULD the same as the value of the <wsa:Action> header block (see Section 5.5: The `<wsa:Action>` Header Block).

Also note that [WSDLv1.1] documents may be defined that specify the value of the SOAPAction header to be included on messages sent to the service defined in WSDL.

4.3. Ordinary ID-* Messages

Ordinary ID- messages* are so-called "application layer" messages or "services" messages, of the forms defined in the Liberty ID-WSF and ID-SIS specification sets or by other applications or services building on the Liberty ID-WSF specifications. These messages as a class are characterized by being able to be correctly conveyed in the "Body" of a SOAP [SOAPv1.1] message. See Example 1. Such messages share the characteristic of needing to be mapped onto an underlying transport or transfer protocol in order for them to be communicated between system entities.

```
<idpp:Query>
  :
  <!--  various message-specific subelements may go here  -->
  :
</idpp:Query>
```

Example 1. A Specific ID-* Message: The `<idpp:Query>` Message

4.4. ID-* Fault Messages

An *ID-* Fault Message* consists of a SOAP `<S:Fault>` element (see Section 3.4: SOAP Fault Types) constructed as specified herein.

When reporting a SOAP processing error such as "`S:VersionMismatch`" or "`S:MustUnderstand`," the `<S:Fault>` element SHOULD be constucted according to [SOAPv1.1].

When reporting a WS-Addressing processing error such as "`wsa:InvalidAddress`," the `<S:Fault>` element SHOULD be constucted according to [WSAv1.0-SOAP].

For all other processing errors the `<S:Fault>` element's attributes and child elements MUST be constrcuted according to these rules:

1. The `<S:Fault>` element:

   A. SHOULD contain a `<faultcode>` element whose value SHOULD be one of "`sbf:FrameworkVersionMismatch`," "`S:server`" or "`S:client`."

   B. SHOULD contain a `<faultstring>` element. This string value MAY be localized.

   C. SHOULD NOT contain a `<S:faultactor>` element.

2. The `<S:Fault>` element's `<detail>` child element SHOULD contain a `<Status>` element (see Section 3.3: Status Types). The `<Status>` element:

   A. MUST contain a `code` attribute set to the value as specified when the issuance of a ID-* Fault message is indicated. Code attribute values defined in this specification are listed above in Section 3.3.1. Other specifications MAY define additional code attribute values.

   B. MAY contain a `ref` attribute set to the value as specified in this specification when the issuance of a ID-* Fault message is indicated.

   C. MAY contain a `comment` attribute set to the value as specified in this specification when the issuance of a ID-* Fault message is indicated. This string value MAY be localized.

3. Additionally, to aid in diagnostics, the header block or message body element referred to by the fault MAY be included in the `<S:Fault>` element's `<detail>` element, after the `<Status>` element.

Note

When reporting SOAP processing errors, the WS-Addressing action http://www.w3.org/2005/08/addressing/soap/fault SHOULD be used. When reporting WS-Addressing processing errors, the WS-Addressing action http://www.w3.org/2005/08/addressing/fault SHOULD be used. When reporting other processing errors, if no specific WS-Addressing action is defined, then http://www.w3.org/2005/08/addressing/soap/fault SHOULD be used.

4.5. SOAP-bound ID-* Messages

ID-* messages are bound into SOAP messages, yielding *SOAP-bound ID-* messages*. This binding thus provides a concrete means for ID-* message conveyance since [SOAPv1.1] specifies a binding to HTTP [RFC2616], which is itself layered onto the ubiquitous [TLS/SSL]/TCP/IP protocol stack.

Although this binding is the only one given in this specification, other protocols could be used to convey ID-* messages, with appropriateness depending on the protocol selected and the target operational context. This is not discussed further in this specification.

A SOAP-bound ID-* message is defined as:

- having all required ID-* header blocks in its `<S:Header>` element, and,
- perhaps having other optional ID-* header blocks in its `<S:Header>` element, and,
- containing either an ordinary ID-* message, or an ID-* fault message, in its `<S:Body>` element. The former is known as an *ordinary SOAP-bound ID-* message* (see Example 2), and the latter is known as a *SOAP-bound ID-* fault message* (see Example 3).

Section 5.11: Messaging Processing Rules specifies the detailed normative processing rules for constructing, sending, and receiving SOAP-bound ID-* messages.

```
<S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:sb="..."
  xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <wsse:Security>
      <wsu:Timestamp>
        <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>

    <wsa:MessageId>...</wsa:MessageId>

    <wsa:To>...</wsa:To>

    <wsa:Action>...</wsa:Action>

    <!-- reference params from target EndpointReference -->

    <sbf:Framework version="2.0"/>

    <sb:Sender providerID="..." affiliationID="..."/>

    ...

  </S:Header>

  <S:Body>

    <idpp:Query> <!-- This is an ID-PP "Query" message bound -->
      :     <!-- into the <S:Body> of a SOAP message.   -->
      :
    </idpp:Query>

  </S:Body>

</S:Envelope>
```

Example 2. An Ordinary SOAP-bound ID-* Message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:sb="..."
  xmlns:pp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <wsse:Security>
      <wsu:Timestamp>
        <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>

    <wsa:MessageId>...</wsa:MessageId>

    <wsa:RelatesTo>...</wsa:RelatesTo>

    <wsa:To>...</wsa:To>

    <wsa:Action>...</wsa:Action>

    <sbf:Framework version="2.0"/>

    <sb:Sender providerID="..."/>

    ...

  </S:Header>

  <S:Body>

    <S:Fault>
      <faultcode>S:server</faultcode>
      <faultstring>Server Error</faultstring>
      <!--  <S:faultactor> should be absent  -->

      <detail>
        <lu:Status code="SomeStatus"
          ref="Foo"
          comment="Bar" />
      </detail>
    </S:Fault>

  </S:Body>

</S:Envelope>
```

Example 3. A SOAP-bound ID-* Fault Message

5. Messaging-specific Header Blocks

This section profiles the use of WS-Addressing SOAP Binding [WSAv1.0-SOAP] and WS-Security [wss-sms] header blocks, as well as defining several new ID-* header blocks, to implement the ID-* message exchange model.

The messaging processing rules associated with the ID-* message exchange model are given in Section 5.11: Messaging Processing Rules.

Additional ID-* header blocks and their processing rules are defined below in Section 6: Optional Header Blocks.

Note

Other ID-* specifications MAY define additional ID-* header blocks.

5.1. The `<wsu:Timestamp>` element in the `<wsse:Security>` Header Block

The `<wsu:Timestamp>` element and the `<wsse:Security>` header block are defined in [wss-sms]. When included in a message, the `<wsu:Timestamp>` element provides a means for the sender to specify the time at which the message was prepared for transmission and the time at which the message should expire.

Note

Depending on the security mechanisms in use [LibertySecMech], it may be necessary to include a `<wsse:Security>` header block solely for the purpose of including the `<wsu:Timestamp>` element.

5.2. The `<wsa:MessageID>` Header Block

The `<wsa:MessageID>` header block is defined in [WSAv1.0-SOAP]. The value of this header block uniquely identifies the message that contains it.

5.2.1. `<wsa:MessageID>` Value Requirements

Values of the `<wsa:MessageID>` header block MUST satisfy the following property:

> Any party that assigns a value to a `<wsa:MessageID>` header block MUST ensure that there is negligible probability that that party or any other party will accidentally assign the same identifier to any other message.

The mechanism by which SOAP-based ID-* senders or receivers ensure that an identifier is unique is left to implementations. In the case that a pseudorandom technique is employed, the above requirement MAY be met by randomly chosing a value 160 bits in length.

Note that [WSAv1.0] requires that `<wsa:MessageID>` values be absolute IRIs.

5.3. The `<wsa:RelatesTo>` Header Block

The `<wsa:RelatesTo>` header block is defined in [WSAv1.0-SOAP]. The value of this header block establishes a relationship between the message that contains it and some other message. The type of relationship is specified in the `RelationshipType` attribute.

Note

When the relationship is http://www.w3.org/2005/03/addressing/reply, the `RelationshipType` attribute may be omitted.

5.4. The `<wsa:To>` Header Block

The `<wsa:To>` header block is defined in [WSAv1.0-SOAP]. The value of this header block specfies the intended destination of the message.

Note

In the typical case that a WS-Addressing endpoint reference is used to address a message, the value of this header block is taken from the `<wsa:Address>` of the endpoint reference. If the `<wsa:To>` header block is not present, the value defaults to http://www.w3.org/2005/03/addressing/role/anonymous; so, when constructing a message, the header block can be omitted if this is the value that would be used. This typically allows the `<wsa:To>` header block to be omitted during synchronous request-response message exchanges over HTTP. Please refer to [WSAv1.0] for default processing rules in the absence of the `<wsa:To>` header block.

5.5. The `<wsa:Action>` Header Block

The `<wsa:Action>` header block is defined in [WSAv1.0-SOAP]. The value of this header block uniquely identifies the semantics implied by the message.

5.6. The `<wsa:ReplyTo>` Header Block

The `<wsa:ReplyTo>` header block is defined in [WSAv1.0-SOAP]. The value of this header block, which is of the WS-Addressing endpoint reference type, specifies the address to which a reply should be sent.

Note

If the <wsa:ReplyTo> header block is not present, the value defaults to http://www.w3.org/2005/03/addressing/role/anonymous; so, when constructing a message, the header block can be omitted if this is the value that would be used. This typically allows the <wsa:ReplyTo> header block to be omitted during synchronous request-response message exchanges over HTTP. Please refer to [WSAv1.0] for default processing rules in the absence of the <wsa:ReplyTo> header block.

5.7. The <wsa:FaultTo> Header Block

The <wsa:FaultTo> header block is defined in [WSAv1.0-SOAP]. The value of this header block, which is of the WS-Addressing endpoint reference type, specifies the address to which a fault should be sent, if one should arise in the processing of the message. If not present, faults are sent to the reply address.

5.8. The <sbf:Framework> Header Block

This section defines the <sbf:Framework> header block. When included in a message, this header provides a means for a sender to communicate the version of the ID-WSF framework used to construct the message.

Framework versions are defined in ID-WSF SCR documents, such as [LibertyIDWSF20SCR].

The schema fragment in Figure 3 defines the <sbf:Framework> header block.

```
<!-- framework header block -->

<xs:complexType name="FrameworkType">
   <xs:sequence>
      <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="version" type="xs:string" use="required"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="Framework" type="FrameworkType"/>
```

Figure 3. The <sbf:Framework> Header Block Schema

```
<sbf:Framework version="2.0" S:mustUnderstand="1"
  S:actor="http://schemas.../next"
  wsu:Id="A72139...381"/>
```

Example 4. An instantiated <sbf:Framework> header block

5.9. The <Sender> Header Block

This section defines the <Sender> header block. When included in a message, this header provides a means for a sender to claim that it is a provider identified by a given providerID value. The sender may also claim that it is a member of a given affiliation. Such claims are generally verifiable by receivers by looking up these values in the sender's metadata [LibertyMetadata].

Note

The providerID claim MAY be used by the receiver as a hint to locate metadata for use in verifying the security of the message (see [LibertyMetadata] and [LibertySecMech]). The mechanisms by which the receiver might locate or establish trust in a provider's metadata are not covered here.

The receiver SHOULD ensure that the claims in the `<Sender>` header block are protected with adequate message security to bind them to the message sender (see [LibertySecMech]).

The `<Sender>` header block defines the following attributes:

- `providerID` [Required] – The Provider ID of the sender.
- `affiliationID` [Optional] – The Affiliation ID of the sender, if any.

The schema fragment in Figure 4 defines the `<Sender>` header block.

```
<!-- sender header block -->

<xs:complexType name="SenderType">
   <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
   <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
 </xs:complexType>

 <xs:element name="Sender" type="SenderType"/>
```

Figure 4. The `<Sender>` Header Block Schema

```
    <sb:Sender S:mustUnderstand="1"
      S:actor="http://schemas.../next"
      wsu:Id="A72139...381"
      providerID="http://example.com"
      affiliationID="http://affiliation.com"/>
```

Example 5. An instantiated `<Sender>` header block

5.10. The `<TargetIdentity>` Header Block

This section defines the `<TargetIdentity>` header block. When included in a message, this header provides a means for the sender to include an *identity token* (see [LibertySecMech]) that specifies an identity at the service that is the target of the message. For example, to obtain profile attributes for a principal, a query message might be sent to a profile service associated with the principal, including an identity token in the target identity header that specifies the principal's identity at the profile service.

Note

If no <TargetIdentity> header block is present, then the invocation identity is typically used as the identity at the service that is the target of the message.

The <TargetIdentity> header is typically only required in cross-principal scenarios such as when one user is accessing the resources of another user.

The <TargetIdentity> header block has a content model of any.

The schema fragment in Figure 5 defines the The <TargetIdentity> header block.

```
<!-- target identity header block -->

<xs:complexType name="TargetIdentityType">
   <xs:sequence>
     <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="TargetIdentity" type="TargetIdentityType"/>
```

Figure 5. The <TargetIdentity> Header Block Schema

```
<sb:TargetIdentity S:mustUnderstand="1"
 S:actor="http://schemas.../next"
 wsu:Id="A31739...293">
 ...
</sb:TargetIdentity>
```

Example 6. An instantiated <TargetIdentity> header block

5.11. Messaging Processing Rules

Overall processing of SOAP-bound ID-* messages follows the rules of the SOAP processing model described in [SOAPv1.1]; specifically, the SOAP mustUnderstand and actor attributes MAY be used to mandate header block processing and target header blocks, respectively. Where applicable, specific processing rules for these attributes are given in the overall processing rules defined below.

The system entity constructing and sending a SOAP-bound ID-* message is called the *sender* in the context of the act of sending the message. The entity receiving this message is called the *receiver* in the context of the act of receiving an individual message (see Section 2.2: Terminology).

Two *Message Exchange Patterns* (MEPs) are supported: one-way, and request-response. One-way is simply where a sender sends a message to a receiver without necessarily expecting to receive an explicit response to the sent message. Request-response is where a sender sends a message to a receiver and expects to receive an explicit response.

The processing rules are described below in terms of Constructing and Sending a SOAP-bound ID-* Message and Receiving and Processing a SOAP-bound ID-* Message . A sender instigating a one-way message exchange will perform only the steps outlined in the former section. A sender participating in a request-response message exchange

will perform the steps in the former section when sending a message, and the steps in the latter section when receiving and processing the response. A receiver participating in a request-response exchange will do the reverse. Note that a receiver of an asynchronous one-way message will perform the steps in the latter section.

Note

The label "ID-* header block(s)" is used to refer to at least one of, or all of, the following set of header blocks:

- `<wsa:MessageID>` [WSAv1.0]
- `<wsa:RelatesTo>` [WSAv1.0]
- `<wsa:To>` [WSAv1.0]
- `<wsa:Action>` [WSAv1.0]
- `<wsa:ReplyTo>` [WSAv1.0]
- `<wsa:FaultTo>` [WSAv1.0]
- `<wsse:Security>` [LibertySecMech]
- `<sbf:Framework>`
- `<Sender>`
- `<TargetIdentity>`
- `<ProcessingContext>`
- `<Consent>`
- `<UsageDirective>`
- `<EndpointUpdate>`
- `<Timeout>`
- `<CredentialsContext>`
- `<ApplicationEPR>`
- `<UserInteraction>`

Other specifications in the Liberty ID-* specification suite MAY define header block(s) not listed above. Nevertheless, they should generally be considered a member of the above list when interpreting the processing rules in this section, and explicitly considered where the processing rules refer to "ID-* header blocks" (see Section 2.2: Terminology).

5.11.1. Constructing and Sending a SOAP-bound ID-* Message

The sender MUST follow these processing rules when constructing and sending an outgoing SOAP-bound ID-* message (hereafter referred to as the *outgoing message*):

1. The outgoing message MUST satisfy the rules given in Section 4: SOAP Binding.
2. The outgoing message MUST satisfy the rules given in [WSAv1.0-SOAP].

3. The outgoing message MUST include exactly one `<wsa:MessageID>` header block in the `<S:Header>` child element of the `<S:Envelope>` element and its value SHOULD be set according to the rules presented in Section 5.2.1: `<wsa:MessageID>` Value Requirements .

4. If the sender is participating in a request-response MEP and is

   A. sending a request message, the outgoing message MUST include at most one `<wsa:ReplyTo>` header block and at most one `<wsa:FaultTo>` header block (if the `<wsa:FaultTo>` header block is not included, faults will be delivered to the reply endpoint)

   B. responding to a prior-received request message, the outgoing message MUST include exactly one `<wsa:RelatesTo>` header block with RelationshipType equal to http://www.w3.org/2005/03/addressing/reply in the `<S:Header>` child element of the `<S:Envelope>` element (note that this is the default RelationshipType and so the attribute MAY be omitted). The value of this header block MUST be set to the value of the `<wsa:MessageID>` header block from the prior-received message.

5. The outgoing message MUST include exactly one `<wsse:Security>` header block. The `<wsse:Security>` header block MUST include a `<wsu:Timestamp>` element. The `<wsu:Timestamp>` element MUST include a `<wsu:Created>` element, the value of which SHOULD be set to the time at which the message is prepared for transmission. This value MUST conform to the rules presented in Section 2.5: Time Values.

   If no clock is available to the message sender then a time value of `1970-01-01T00:00:00Z` SHOULD be used.

6. The sender MUST include exactly one `<sbf:Framework>` header block in the `<S:Header>` child element of the `<S:Envelope>` element. The `version` attribute of this `<sbf:Framework>` header element MUST be set to ID-WSF version in use by the sender.

7. If the sender is acting in the role of a Liberty provider, the message MUST include exactly one `<Sender>` header block in the `<S:Header>` child element of the `<S:Envelope>` element. The attributes of this `<Sender>` header block MUST be set as follows:

   A. `providerID` MUST be present and SHOULD be set to a value appropriate for the sender to claim [LibertyMetadata].

   B. `affiliationID` MAY be present. If so, it SHOULD be set to a value appropriate for the sender to claim [LibertyMetadata].

8. The sender MAY include a `<TargetIdentity>` header block, as needed, to identify the target identity of the message. The sender MUST NOT include more than one `<TargetIdentity>` header block.

9. The sender MAY include other ID-* header blocks in the message, in addition to those enumerated above, as required by the overall messaging and processing context. For example, the sender may include a `<wsse:Security>` header block [LibertySecMech].

10. The sender adds either:

    A. an ordinary ID-* message (as described in Section 4.3: Ordinary ID-* Messages; see Example 2), or,

    B. an ID-* fault message (as **prescribed** in Section 4.4: ID-* Fault Messages; see Example 3),

    to the SOAP-bound ID-* message's `<S:Body>` element.

11. The sender also performs any needed additional preparation of the message, for example including other header blocks, and signing some or all of the message elements, and then sends the message to the receiver. See Section 5.12: Examples .

5.11.2. Receiving and Processing a SOAP-bound ID-* Message

The receiver of a SOAP-bound ID-* message, either ordinary or fault, MUST perform the following processing steps on the ID-* header blocks of the incoming SOAP-bound ID-* message.

Note

Although the steps below are explicitly arranged and numbered sequentially, the intent is **not** to strictly define a specific overall processing algorithm in terms of having implementations follow these steps in exactly the same sequence on a per-header-block basis. However, all specified tests MUST be applied as appropriate to all ID-* header blocks in the incoming SOAP-bound ID-* message.

1. The incoming message MUST satisfy the rules given in Section 4: SOAP Binding.

2. The incoming message MUST satisfy the rules given in [WSAv1.0-SOAP].

3. Processing specific to the `<sbf:Framework>` header block:

   A. A single `<sbf:Framework>` header block MUST be present in the header of the message.

   B. The value of the `version` attribute of the `<sbf:Framework>` header element MUST specify an ID-WSF version supported by the receiver. Further processing MUST be according to the processing rules of the specifieded version.

   C. If the foregoing test (3.A) holds true, processing continues with step 4 .

   D. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4) with the `<faultcode>` of `sbf:FrameworkVersionMismatch`.

   The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

4. Processing specific to the `<wsu:Timestamp>` element in the `<wsse:Security>` header block:

   A. A single `<wsse:Security>` header block MUST be present in the header of the message. The `<wsse:Security>` header block MUST include a `<wsu:Timestamp>` element. The `<wsu:Timestamp>` element MUST include a `<wsu:Created>` element.

   B. The value of the `<wsu:Created>` element SHOULD be within an appropriate offset from local time expressed in UTC. Absent other guidance, a value of 5 minutes MAY be used.

   If the `<wsu:Timestamp>` element includes an `<wsu:Expires>` element, the time at the receiver MUST be before that time.

Note

Certain classes of client devices, such as consumer electronics, often do not have correctly set clocks. These processing rules may be relaxed for messages received from such devices.

C. If the foregoing test (4.A) holds true, processing continues with step 5 .

D. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4) with the `<Status>` element configured with:

- a `code` attribute with a value of:
  - `"IDStarMsgNotUnderstood"` if the failed test is 4.A.
  - `"StaleMsg"` if the failed test is 4.B,
- and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

The `<S:Fault>` SHOULD contain a `<faultcode>` of `S:Client`.

The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

Note

This specification does not include specific processing rules designed to ensure reliable message delivery or to prevent message replay. Services building on this specification should expect that clients may re-transmit messages for which no reply has been received.

5. Processing specific to the `<wsa:MessageID>` and `<wsa:RelatesTo>` header blocks:

A. A single `<wsa:MessageID>` header block MUST be present in the header of the message.

B. If the `<wsa:RelatesTo>` header block with RelationshipType equal to http://www.w3.org/2005/03/ addressing/reply is present, and if the receiver is participating in a request-response MEP with the sending party, then the value of the `<wsa:RelatesTo>` header block SHOULD match the value of the `<wsa:MessageID>` header block of a message previously sent by the receiver to the sender of the now incoming message.

C. If the foregoing tests (5.A through 5.B) hold true, processing continues with step 6 .

D. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4).

The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

Note

This specification does not include specific processing rules designed to ensure reliable message delivery or to prevent message replay. Services building on this specification should expect that clients may re-transmit messages for which no reply has been received.

6. At this point, the receiver of the message MAY cease processing the message and indicate to the sender that the message should be re-submitted to a different endpoint, according to the rules specified in Section 6.4.5.1

7. Processing specific to the `<Sender>` header block:

   A. Verify that any declared `providerID` or `affiliationID`, are valid. The receiver SHOULD perform this verification and validation against metadata (see [LibertyMetadata]).

   The declared providerID and affiliationID MUST NOT be used to establish a security context for further processing of the message on their own, but must be validated by an adequate security mechanism as specified in [LibertySecMech].

   B. If the foregoing test (7.A) holds true, processing continues with step 8.

   C. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4) with the `<Status>` element configured with:

   - a `code` attribute with a value of:
     - `"ProviderIDNotValid"`, or,
     - `"AffiliationIDNotValid"`, as appropriate (if both the claimed Provider ID and the Affiliation ID
       are deemed invalid, then the returned code SHOULD be `"AffiliationIDNotValid"`),
   - and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

   The `<S:Fault>` SHOULD contain a `<faultcode>` of `S:Client`.

   The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

8. Processing specific to the `<TargetIdentity>` header block:

   A. Verify that any provided target identity token is valid (see [LibertySecMech]) and, if appropriate, that the identity specified by the token is known.

   B. If the foregoing test (8.A) holds true, processing continues with step 9.

   C. Otherwise, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4) with the `<Status>` element configured with:

   - a `code` attribute with a value of:
     - `"TargetIdentityNotValid"`
   - and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

   The `<S:Fault>` SHOULD contain a `<faultcode>` of `S:Client`.

   The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

9. All remaining ID-* header blocks SHOULD be processed at this point. See appropriate sections in this and other specifications for the processing rules associated with these header blocks and the manner of reporting any issues with this processing. If there are no issues with these header blocks, then processing continues with step 10 below, otherwise the receiver is finished processing this incoming message at this point.

Note

It should be noted that the receiver MAY return an `InappropriateCredentials` based on their processing of the `<wsse:Security>` header block, under conditions specified below in Section 6.4 and Section 6.3, in addition to other conditions such as an expired credential (see [LibertySecMech]).

10. If the incoming message's applicable header blocks have passed all specified and applicable tests, the incoming message SHOULD be dispatched for further processing as appropriate.

    If the message contained in the encompassing SOAP message's `<S:Body>` element is not dispatchable, the receiver MAY respond to the sender with a SOAP-bound ID-* Fault message (per Section 4.4) with the `<Status>` element configured with:

    - a `code` attribute with a value of:
      - `"IDStarMsgNotUnderstood"`
    - and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

    Receivers MUST be able to avoid ID-* fault message "loops." For example, if the incoming message is conveying an ID-* fault message, and there is some issue with one or more of its ID-* header blocks, the receiver should not issue a SOAP-bound ID-* Fault message in response.

Note

Other specifications conforming to this binding that specify ordinary ID-* messages and their processing, such as [LibertyIDPP] or [LibertyDisco], MAY define <Status> element code attribute values in addition to the ones defined in Section 3.3.1 of this document. These code attribute values SHOULD be used to signal to the sender any issues with the incoming ordinary ID-* message found by the receiver. This specification does not define any such conditions other than the one described above in 10, and they are not further discussed in this document.

5.12. Examples

Example 7 illustrates a SOAP-bound ID-* message conveying a Personal Profile (ID-PP) Modify request message [LibertyIDPP].

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:sbf="urn:liberty:sb"
  xmlns:sb="urn:liberty:sb:2006-08"
  xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <wsse:Security>
      <wsu:Timestamp>
        <wsu:Created>2005-06-17T04:49:17Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>

    <wsa:MessageID>
       http://spwsc.com/0123456789abcdef0123456789abcdef01234567
    </wsa:MessageID>

    <wsa:To>http://spwsp.com/idpp</wsa:To>

    <wsa:Action>urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>

    <!-- reference params from target EndpointReference -->

    <sbf:Framework version="2.0"/>

    <sb:Sender providerID="http://spwsc.com" affiliationID="http://affiliation.com"/>

    ...

  </S:Header>

  <S:Body>

    <idpp:Modify>
      :     <!-- this is an ID-PP Modify message -->
    </idpp:Modify>

  </S:Body>

</S:Envelope>
```

Example 7. A SOAP-bound ID-* Request Message

Example 8 illustrates a SOAP-bound ID-* response to the message in the previous example, which conveyed an ID-PP Modify message. Note how the `<wsa:RelatesTo>` header value references the `<wsa:MessageID>` in the example above.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:sbf="urn:liberty:sb"
  xmlns:sb="urn:liberty:sb:2006-08"
  xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <wsse:Security>
      <wsu:Timestamp>
        <wsu:Created>2005-06-17T04:49:18Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>

    <wsa:MessageID>
       http://spwsp.com/ffeeddccbbaa99887766554433221100ffeebbcc
    </wsa:MessageID>
    <wsa:RelatesTo>
       http://spwsc.com/0123456789abcdef0123456789abcdef01234567
    </wsa:RelatesTo>

    <wsa:Action>urn:liberty:id-sis-pp:2003-08:ModifyResponse</wsa:Action>

    <sbf:Framework version="2.0"/>

    <sb:Sender providerID="http://spwsp.com" />

    ...

  </S:Header>

  <S:Body>

    <idpp:ModifyResponse>
      :     <!-- this is an ID-PP ModifyResponse message -->
    </idpp:ModifyResponse>

  </S:Body>

</S:Envelope>
```

Example 8. A SOAP-bound ID-* Response Message

6. Optional Header Blocks

The optional header blocks described in this specification are:

- `<ProcessingContext>`
- `<Consent>`
- `<CredentialsContext>`
- `<EndpointUpdate>`
- `<Timeout>`
- `<UsageDirective>`
- `<ApplicationEPR>`
- `<UserInteraction>`

The following sections describe these optional ID-* header blocks along with their specific processing rules.

Note

Whenever an optional header block appears in a SOAP-bound ID-* message, the processing rules specific to that header block (which are given in this section, below) MUST be used in combination with the messaging processing rules given above in Section 5.11: Messaging Processing Rules. This applies whether the message is being constructed and sent, or being received and processed.

6.1. The `<ProcessingContext>` Header Block

This section defines the `<ProcessingContext>` header block. This header block may be employed by a sender to signal to a receiver that the latter should add a specific additional facet to the overall *processing context* in which any action(s) are invoked as a result of processing any ID-* message also conveyed in the overall SOAP-bound ID-* message. The full semantics of this header block are described below in Section 6.1.2: `<ProcessingContext>` Header Block Semantics and Processing Rules .

Processing context facets are denoted by URIs. URIs are assigned to denote specific processing context facets. This specification defines several such URIs below in Section 6.1.2.2.

6.1.1. The `<ProcessingContext>` Type and Element

The `<ProcessingContext>` content model is `anyURI`.

The schema fragment in Figure 6 defines the `<ProcessingContext>` header block.

```
<!-- processing context header block -->

<xs:complexType name="ProcessingContextType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="ProcessingContext" type="ProcessingContextType"/>
```

Figure 6. The <ProcessingContext> Header Block Schema

```
<sb:ProcessingContext S:mustUnderstand="1">
   urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline
</sb:ProcessingContext>
```

Example 9. An instantiated <ProcessingContext> header block

6.1.2. <ProcessingContext> Header Block Semantics and Processing Rules

This section first describes the overall semantics of the <ProcessingContext> header block, then defines two *processing context facet URIs*, and concludes with defining specific processing rules.

6.1.2.1. <ProcessingContext> Header Block Semantics

The overall semantic of the <ProcessingContext> header block is:

> The <ProcessingContext> header block MAY be employed by a sender, who is acting in a web services client (WSC) role, to signal to a receiver, who is acting in a web services provider (WSP) role, that the latter should add a specific *processing context facet* to the overall *processing context* (see Section 2.2: Terminology) in which the *service request* is evaluated.

The specific processing context facet being conveyed by the <ProcessingContext> header block is identified by the header block's URI element value.

Such URIs are known as *processing context facet URIs*. An example of a processing context facet that may be signaled by such a URI is whether the principal should be considered to be online or not.

An ID-WSF or ID-SIS WSP receiving a service request containing a <ProcessingContext> header block with one of the above processing context facet URIs SHOULD process the conveyed ID-* message **with the indicated processing context facet in force**. Thus the ID-* message's processing as well as any applicable access management policies are exercised within an overall processing context which includes the processing context facet. Finally, an indication of success or failure of the ID-* message processing is returned to the sender, in the same manner as would be done if the ID-* message had been sent without the attendant <ProcessingContext> header block.

The above completely describes the semantic of this header block itself, and further description of particular effects on processing must be made in descriptions of processing context facet URIs. Such a description is given in the next section.

Note

Whether or not a receiver honors a `<ProcessingContext>` header block is a matter of local policy at the receiver, as is whether or not a receiver honors any given request from any given sender. For example, the `<ProcessingContext>` header block functionality has security implications in the sense of possibly facilitating an adversary to probe a receiver's behavior given adversary-chosen inputs. For these reasons, whether or not the `<ProcessingContext>` header block functionality is enabled on the part of a receiver with respect to a particular sender should be a matter of business-level agreement between the receiver and the sender.

6.1.2.2. Processing Context Facet URIs: PrincipalOnline, PrincipalOffline, and Simulate

Three processing context facet URIs are defined below for use with the `<ProcessingContext>` header block:

`urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline`
: Conduct the processing of the ID-* message as if the Principal is offline.

`urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline`
: Conduct the processing of the ID-* message as if the Principal is online.

`urn:liberty:sb:2003-08:ProcessingContext:Simulate`
: *Simulate* the processing of the ID-* message.

If the sender includes a `<UserInteraction>` header block in addition to the `<ProcessingContext>` header block in the SOAP-bound ID-* request message, the receiver and sender MUST appropriately initiate the indicated user interaction, and incorporate information supplied by the user as a part of the resultant user interaction, into the appropriate data and/or policy stores.

Note

Any processing context facet that was conveyed in the `<ProcessingContext>` header block MUST NOT be enforced during such a user interaction. Rather, it applies only to the processing of the ID-* message itself.

In summary, the overall intended side-effect of using the above-defined processing context facets is for the receiver to evaluate applicable policy, and return a putative indication of success or failure to the sender. This provides WSCs the capability to make an ID-WSF or ID-SIS service request and ascertain whether it will be successful or not–without the service request actually being carried out. Additionally, it facilitates carrying out any user interaction that may be indicated by the current combination of service request context and applicable policy. This will, for example, facilitate some WSCs to fashion more "user friendly" experiences.

6.1.2.3. Defining New Processing Context Facet URIs

The rightmost portions of the processing context facet URIs after the "ProcessingContext:" component are referred to as *processing context facet identifiers*. For example, whether the Principal is online or not is a facet of a request context. New processing context facet identifiers MAY be defined in other specifications, for example in ID-SIS data service specifications. An ID-SIS data service may define as many levels of request context identifiers as necessary to address the application's needs.

6.1.2.4. Sender Processing Rules

A sender MAY include a `<ProcessingContext>` header block in a SOAP-bound ID-* message. The sender MUST include a processing context facet URI in the `<ProcessingContext>` header block. The sender then sends the ID-* SOAP-bound message to an ID-WSF or ID-SIS service-hosting node (AKA the receiver).

A sender MAY indicate that it believes either that the Principal is currently "online" or "offline" when it sends a message by specifying one of the two processing context facet URIs:

- `urn:liberty:sb:2003-08:ProcessingContext:PrincipalOnline`
- `urn:liberty:sb:2003-08:ProcessingContext:PrincipalOffline`

The sender will typically receive a response from the receiver indicating success or failure or will receive a SOAP fault indicating a processing error with the SOAP-bound ID-* message. Note that in the case of a "successful" request *simulation*, the service will not return any result data other than an indication of success or failure to the sender.

6.1.2.5. Receiver Processing Rules

The receiver of a request containing a `<ProcessingContext>` header block MUST examine the included processing context facet URI. If it is known to the data service, then the data service MUST attempt to process the data service request, represented by the ID-* message, in an overall processing context including the processing context facet as indicated by the conveyed processing context facet URI, and return an indication of success or failure to the sender.

If the data service request is malformed or has some other issue that would normally cause the receiver to issue a SOAP fault, the receiver SHOULD do so.

If the receiver is asked to simulate processing of the request (by the inclusion of the `urn:liberty:sb:2003-08:ProcessingContext:Simulate` facet URI), and they are both able and willing to honor that processing context, then the receiver MUST evaluate the conveyed ID-* message, but MUST NOT actually perform the operation. That is, the receiver MUST NOT make actual changes to underlying ID-* service datastore, and it MUST NOT return any data as a result of evaluating the ID-* message.

If the sender includes a `<UserInteraction>` header block, in addition to the `<ProcessingContext>` header block, then both participants MUST initiate the indicated user interaction appropriately, and incorporate information supplied by the user as a part of the interaction into appropriate data and/or policy stores, even if the `urn:liberty:sb:2003-08:ProcessingContext:Simulate` URI is specified in a `<ProcessingContext>` header.

In the event the receiver does not understand the included processing context facet URI, the receiver MAY respond with a SOAP-bound ID-* fault message (per Section 4.4: ID-* Fault Messages) with the `<Status>` element configured with:

- a `code` attribute with a value of:
  - `"ProcCtxURINotUnderstood"`
- and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

In the event the receiver is not willing to enforce a stipulated processing context, the receiver MAY respond with a SOAP-bound ID-* fault message (per Section 4.4: ID-* Fault Messages) with the `<Status>` element configured with:

- a `code` attribute with a value of:
  - `"ProcCtxUnwilling"`
- and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

Note

The receiver MAY reference multiple `<ProcessingContext>` headers in the `<detail>` of the fault response (in accordance with the rules specified in Section 4.4).

6.2. The `<Consent>` Header Block

This section defines the `<Consent>` header block. This header block is used to explicitly claim that the Principal consented to the present interaction.

6.2.1. The `<Consent>` Type and Element

The `<Consent>` header block element MAY be employed by either a sender or a receiver. For example, the Principal may be using a Liberty-enabled client or proxy (common in the wireless world), and in that sort of environment the mobile operator may cause the Principal's terminal (AKA: cell phone) to prompt the principal for consent for some interaction.

The `<Consent>` header block has the following attributes:

- `uri` [Required] – A URI indicating that the Principal's consent was obtained.

  Optionally, the URI MAY identify a particular *Consent Agreement Statement* defining the specific nature of the consent obtained.

  This specification defines one well-known URI Liberty implementors and deployers MAY use to indicate positive Principal consent was obtained with respect to whatever ID-* interaction is underway or being initiated. This URI is known as the "Principal Consent Obtained" URI (PCO). The value of this URI is:

  `urn:liberty:consent:obtained`

  This URI does not correspond to any particular Consent Agreement Statement. Rather, it simply states that consent was obtained. The full meaning and implication of this will need to be derived from the execution context.

- `timestamp` [Optional] – For denoting the time at which the sender obtained Principal consent with the POC.

The schema fragment in Figure 7 defines the `Consent` header block type.

```
<!-- consent header block -->

<xs:complexType name="ConsentType">
  <xs:attribute name="uri" type="xs:anyURI" use="required"/>
  <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="Consent" type="ConsentType"/>
```

Figure 7. The `<Consent>` Header Block Schema

```
<sb:Consent
   uri="urn:liberty:consent:obtained"
   timestamp="2112-03-15T11:12:10Z"/>
```

Example 10. An instantiated `<Consent>` header block

6.3. The <CredentialsContext> Header Block

6.3.1. Overview

It may be necessary for an entity receiving an ID-* message to indicate the type of credentials that should be used by the sender in submitting a message.

6.3.2. CredentialsContext Type and Element

Receivers of an ID-* message MAY add <CredentialsContext> elements to the SOAP header of their response.

The element is based upon the CredentialsContextType which is defined as:

- samlp:RequestedAuthnContext [Optional] – a container that allows the expression of a requested authentication context (see [SAMLCore2]).
- SecurityMechID [Optional] – A set of elements that specify ID-WSF security mechanism URIs (see [Liberty-SecMech]).

The following schema fragment describes the <CredentialsContext> header block.

```
<!-- credentials context header block -->

<xs:complexType name="CredentialsContextType">
   <xs:sequence>
      <xs:element ref="samlp:RequestedAuthnContext" minOccurs="0"/>
      <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="CredentialsContext" type="CredentialsContextType"/>
```

Figure 8. The <CredentialsContext> Header Block Schema

6.3.3. CredentialsContext Example

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:sb="urn:liberty:sb:2006-08"
        xmlns:lib="urn:liberty:iff:2003-08">

   <S:Header>
      ...
      <!-- Says that the sender would like credentials that include RequestAuthnContext
          as specified -->

      <sb:CredentialsContext mustUnderstand="1">

         <samlp:RequestedAuthnContext>
           ...
         </samlp:RequestedAuthnContext>

      </sb:CredentialsContext>
      ...
   </S:Header>

   <S:Body>

      <!-- a fault in the body indicates that the WSP's policy requires
      different (perhaps "stronger") credentials than were originally
      provided in the request -->

      <S:Fault>
         <faultcode>S:Server</faultcode>
         <faultstring>
            Your request contained inappropriate credentials.
         </faultstring>
         <detail>
            <lu:Status code="InappropriateCredentials"/>
            <wsse:Security id="a6352...564"/>
         </detail>
      </S:Fault>
   </S:Body>

</S:Envelope>
```

Example 11. A CredentialsContext Header Offered in Response to a Request with Inappropriate Credentials.

6.3.4. Processing Rules

6.3.4.1. Sender Processing Rules

A sender including this header MUST specify at least one `RequestAuthnContext` or one `SecurityMechID`.

The `SecurityMechID` elements SHOULD be listed in order of preference by the sender.

6.3.4.2. Receiver Processing Rules

The receiver of a `<CredentialsContext>` header containing one or more `SecurityMechID` elements SHOULD use the highest-listed (first) `SecurityMechID` that it supports in future requests to the sender of this header.

The receiver of a `<CredentialsContext>` header containing a `RequestAuthnContext` element SHOULD use credentials that conform to the policies specified therein in any future requests to the sender of this header (where credentials are required).

6.4. The `<EndpointUpdate>` Header Block

6.4.1. Overview

It may be necessary for an entity receiving an ID-* message to indicate that messages from the sender should be directed to a different endpoint, or that they wish a different credential to be used than was originally specified by the entity for access to the requested resource. The `<EndpointUpdate>` response header allows a message receiver to indicate that a new endpoint or new credentials should be employed by the sender of the message on any subsequent messages. This header block may be used in conjunction with the `<sb:InappropriateCredentials>` and `<sb:EndpointUpdated>` faults, to indicate that the current message processing failed for those reasons, and should be submitted with the changes noted in any accompanying `<EndpointUpdate>` header block.

Note

The use of this header block allows the sender of the message to convey updates to security tokens, essentially providing a token renewal mechanism. This is not discussed further in this specification.

6.4.2. `EndpointUpdate` Type and Element

Receivers of an ID-* message may add an `<EndpointUpdate>` element to the SOAP header of their response.

This element is based upon the `EndpointUpdateType` which is an extension of `wsa:EndpointReferenceType` that adds the following attributes:

- `updateType` [Optional] – A URI attribute indicating whether the update should be interpreted as completely superseding the endpoint reference used to send the current request (the default) or whether it should be interpreted as a partial updated.

  `urn:liberty:sb:2006-08:EndpointUpdate:Complete`
  A complete update.

  `urn:liberty:sb:2006-08:EndpointUpdate:Partial`
  A partial update. The complete updated endpoint reference is contructed according to the processing rules below.

The following schema fragment describes the `<EndpointUpdate>` header block.

```
<!-- epr update header block -->

<xs:complexType name="EndpointUpdateType">
   <xs:complexContent>
      <xs:extension base="wsa:EndpointReferenceType">
         <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:element name="EndpointUpdate" type="EndpointUpdateType"/>
```

Figure 9. The `<EndpointUpdate>` Header Block Schema

6.4.3. EndpointUpdate Examples

1. Service responds to a request, indicating a new security mechanism and credential

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:sb="urn:liberty:sb:2006-08"
        xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
      <wsa:Address>urn:liberty:sb:2006-08:EndpointUpdate:NoChange</wsa:Address>
      <wsa:Metadata>
        <ds:SecurityContext>
          <ds:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</sb:SecurityMechID>
          <wsse:SecurityTokenReference>
            <wsse:Embedded>
              <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="..."
                 ValueType="anyNSprefix:ServiceSessionContext">
                ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDNhMzc3Nwo=
              </wsse:BinarySecurityToken>
            </wsse:Embedded>
          </wsse:SecurityTokenReference>
        </ds:SecurityContext>
      </wsa:Metadata>
    </sb:EndpointUpdate>

    ...

  </S:Header>

  <S:Body>

    <idpp:QueryResponse>
      ...
    </idpp:QueryResponse>

  </S:Body>

</S:Envelope>
```

```
    2. The client sends a new request, using the contents of the EndpointUpdate

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <wsse:Security xmlns:wsse="...">

      <wsse:BinarySecurityToken xmlns:wsse="..." wsu:Id="bst"
         ValueType="anyNSprefix:ServiceSessionContext">
        ZjgzOWZlNzgyZTk1ZWU3OWEyMTRlODVmNGZkYzE4MmQ2ZDNhMzc3Nwo=
      </wsse:BinarySecurityToken>

    </wsse:Security>

    ...

  </S:Header>

  <S:Body>

    <idpp:Query>
      ...
    </idpp:Query>

  </S:Body>

</S:Envelope>
```

Example 12. An EndpointUpdate Specifying a ServiceSessionContext Token, and the TLS Bearer Security Mechanism.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:sb="urn:liberty:sb:2006-08">

  <S:Header>

    ...

    <sb:EndpointUpdate mustUnderstand="1" updateType="urn:liberty:sb:2004-04:Partial">
      <wsa:Address>http://example.com:443/soap</wsa:Address>
    </sb:EndpointUpdate>

    ...

  </S:Header>

  <S:Body>

    <!-- a fault in the body indicates that the request corresponding
    to this response should be re-submitted to the endpoint -->

    <S:Fault>

      <faultcode>S:Server</faultcode>

      <faultstring>
        You must resubmit this request to the new endpoint.
      </faultstring>

      <detail>
        <lu:Status code="EndpointUpdated"/>
      </detail>

    </S:Fault>

  </S:Body>

</S:Envelope>
```

Example 13. An EndpointUpdate Specifying an Updated Address.

6.4.4. Processing Rules for the EndpointUpdate header

6.4.4.1. Sender Processing Rules

The receiver of an ID-* message MAY add an `<EndpointUpdate>` header block to their response.

If `updateType` is not present or has the value `urn:liberty:sb:2006-08:EndpointUpdate:Complete`, the `<wsa:EndpointUpdate>` MUST be a completely specified endpoint reference.

If `updateType` has the value `urn:liberty:sb:2006-08:EndpointUpdate:Partial`, the `<wsa:EndpointUpdate>` MAY omit any direct children of `<wsa:RefenceParameters>` or `<wsa:Metadata>` that have not changed from the original endpoint reference used to send the current request. Similarly, any extension elements that have not changed MAY be omitted. If the address has not changed, then the URI

`urn:liberty:sb:2006-08:EndpointUpdate:NoChange`

MAY be used in the `<wsa:Address>` value to indicate that the original address should continue to be used.

Note

The expressiveness of partial updates is limited. In particular, updates to `<wsa:ReferenceParameters>` and `<wsa:Metadata>` are done based on the qualified names of the direct children of those containers. If any child with a matching name is provided in the update, then all children with that name in the original are replaced. It is also impossible, with a partial update, to remove an element; elements may only be added or replaced.

6.4.4.2. Receiver Processing Rules

The receiver of an `<EndpointUpdate>` header SHOULD use the specified endpoint reference values to address any future requests to the sender of the header (where the endpoint reference used to address the request that resulted in the response containing the header would have been used), until newer information is obtained through this or some other mechanism or the updated information expires. If the updated information has a shorter lifetime than the current information (that it updates), then the current information SHOULD be retained as a fallback for when the updated information expires.

If `updateType` is not present or has the value `urn:liberty:sb:2006-08:EndpointUpdate:Complete`, the `<wsa:EndpointUpdate>` is a completely specified endpoint reference.

If `updateType` has the value `urn:liberty:sb:2006-08:EndpointUpdate:Partial`, the `<wsa:EndpointUpdate>` is a partially specified endpoint reference. The following steps are used to construct a complete endpoint reference from the endpoint reference that was used to address the request that resulted in the response containing this header:

1. Take the `<wsa:Address>` from the `<wsa:EndpointUpdate>`. If the value is `urn:liberty:sb:2006-08:EndpointUpdate:NoChange`, then take the `<wsa:Address>` from the original endpoint reference.

2. Take the `<wsa:ReferenceParameters>` from the `<wsa:EndpointUpdate>`, if present. Then, if `<wsa:ReferenceParameters>` is present in the orginal endpoint reference, take each direct child from that element that does not match an element already taken from the update (comparing the namespace qualified names of the elements).

3. Take the `<wsa:Metadata>` from the `<wsa:EndpointUpdate>`, if present. Then, if `<wsa:Metadata>` is present in the orginal endpoint reference, take each direct child from that element that does not match an element already taken from the update (comparing the namespace qualified names of the elements).

4. Take any extension elements from the `<wsa:EndpointUpdate>`, if present. Then, if any extension elements are present in the orginal endpoint reference, take each one that does not match an element already taken from the update (comparing the namespace qualified names of the elements).

6.4.5. Processing Rules for the EndpointUpdated SOAP Fault

6.4.5.1. Sender Processing Rules

The receiver of an ID-* message MAY issue a SOAP Fault indicating that the endpoint to which this message was submitted has permanently changed.

Once the receiver has sent this fault response, no further processing of the message should take place.

If the receiver chooses to send the fault response, then it SHOULD also include an `<EndpointUpdate>` header, indicating the new endpoint which should be used to re-submit this message, and any further messages directed to the responding service.

6.4.5.2. Receiver Processing Rules

If the receiver of this fault response also received an <EndpointUpdate> header in the response, it MAY re-submit the failed request to any endpoint specified in that header, but it SHOULD provide a different <wsa:MessageID> header block value in the request.

6.5. The <Timeout> Header Block

6.5.1. Overview

A requesting entity may wish to indicate that they would like a request to be processed within some specified amount of time. Such an entity would indicate their wish via the <Timeout> header block.

6.5.2. Timeout Type and Element

Senders of ID-* messages MAY add a <Timeout> element to the SOAP header of their request.

This element is based upon the TimeoutType which is defined as:

- maxProcessingTime [Required] – an integer specifying (in seconds) the maximum amount of time the sender wishes the receiver to spend in processing their request

The following schema fragment describes the <Timeout> header block.

```
<!-- timeout header block -->

<xs:complexType name="TimeoutType">
   <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="Timeout" type="TimeoutType"/>
```

Figure 10. The <Timeout> Header Block Schema

6.5.3. Timeout Example

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:sb="urn:liberty:sb:2006-08"
        xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <sb:Timeout mustUnderstand="1" wsu:Id="timeout.123"
             maxProcessingTime="7"/>

    ...

  </S:Header>

  <S:Body>

    <idpp:Query>
      ...
    </idpp:Query>

  </S:Body>

</S:Envelope>
```

Example 14. Example of a Request with Timeout Specified

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:sb="urn:liberty:sb:2006-08">

  <S:Header>

    ...

  </S:Header>

  <S:Body>

    <S:Fault>

      <faultcode>
        S:Server
      </faultcode>

      <detail>

        <lu:Status code="ProcessingTimeout"/>

        <!-- Reference the specified Timeout header, if it was supplied
            by the sender -->

        <sb:Timeout wsu:Id="timeout.123"/>

      </detail>
    </S:Fault>

  </S:Body>

</S:Envelope>
```

Example 15. Example of a Timed-out Response

6.5.4. Processing Rules

6.5.4.1. Receiver Processing Rules

The receiver of a `<Timeout>` header SHOULD NOT begin processing of a message (beyond processing of the SOAP headers as noted in this specification) if it expects that such processing would exceed the value specified in the maxProcessingTime attribute.

The receiver MUST respond to the message within the number of seconds specified in the maxProcessingTime attribute.

If the receiver is unable to complete processing within the number of seconds specified in the `maxProcessingTime` attribute of the `<Timeout>` header, then they MUST respond with a SOAP Fault with a `code` of `ProcessingTimeout`.

Note

If the sender of a message does not include a `<Timeout>` header, but the receiver wishes to indicate to the sender that server processing failed due to a timeout, then the receiver MAY respond with a SOAP Fault with a `code` of `ProcessingTimeout`.

6.6. The `<UsageDirective>` Header Block

This section defines the ID-* usage directive facilities.

6.6.1. Overview

Participants in the ID-WSF framework may need to indicate the privacy policy associated with a message. To facilitate this, senders, acting as either a client or a server, may add one or more `<UsageDirective>` header blocks to the SOAP Header of the message being sent. A `<UsageDirective>` appearing in a SOAP-based ID-* request message expresses *intended usage*. A `<UsageDirective>` appearing in a response expresses *how* the receiver of the response is to use the response data. A `<UsageDirective>` in a response message containing no ID-WSF response message data, a fault response for example, may be used to express policies acceptable to the responder.

6.6.2. `UsageDirective` Type and Element

Senders MAY add a `<UsageDirective>` element to the SOAP header. This element is based upon the `UsageDirectiveType` which is defined as:

- `ref` [Required] – An attribute referring to an element of the SOAP-based ID-* message to which the usage directive applies.
- `<element>(s)` [Optional] – Elements, comprising an instance of some policy expression language, whose purpose is to express the actual policy the usage directive is conveying. The `ref` attribute above points at the element in the overall SOAP-based ID-* message to which the usage directive applies.

The schema fragment in Figure 11 defines the <UsageDirective> header type and element.

```
<!-- usage directive header block -->

<xs:complexType name="UsageDirectiveType">
  <xs:sequence>
    <xs:any namespace="##other" processContents="lax"
      maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:IDREF" use="required"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="UsageDirective" type="UsageDirectiveType"/>
```

Figure 11. The <UsageDirective> Header Block Schema

6.6.3. Usage Directive Examples

Example 16 illustrates a SOAP-based ID-* message, containing a <UsageDirective> header block, and conveying a Personal Profile (ID-PP) Modify message [LibertyIDPP]. The <UsageDirective> header block contains a usage directive expressed in a policy language identified by the cot: namespace and the URI http://cot.example.com/policies/eu-compliant, and applying to the ID-PP Query message identified by the id of datarequest001.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:sb="urn:liberty:sb:2006-08"
  xmlns:pp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    ...

    <sb:UsageDirective S:mustUnderstand="1"
      ref="#datarequest001">

      <cot:PrivacyPolicyReference
        xmlns:cot="http://cot.example.com/isf">
        http://cot.example.com/policies/eu-compliant
      </cot:PrivacyPolicyReference>

    </sb:UsageDirective>

    ...

  </S:Header>

  <S:Body>

    <pp:Query id="datarequest001" xmlns:pp="urn:liberty:id-sis-pp:2003-08">
      <pp:ResourceID>data:d8ddw6dd7m28v628</pp:ResourceID>
      <pp:QueryItem>
        <pp:Select>/pp:IDPP/pp:IDPPAddressCard</pp:Select>
      </pp:QueryItem>
    </pp:Query>

  </S:Body>

</S:Envelope>
```

Example 16. A Usage Directive on a Request for the Address of a Principal.

6.6.4. Processing Rules

6.6.4.1. Sender Processing Rules

The sender of a SOAP-based ID-* message with a `<UsageDirective>` header block MUST ensure that the value of the `ref` attribute is set to the value of the `id` of the appropriate element in the message. The sender SHOULD ensure that the `<UsageDirective>` is integrity-protected. The protection mechanism, if utilized, SHOULD be in accordance with those defined in [LibertySecMech].

6.6.4.2. Receiver Processing Rules

A receiver of a SOAP-based ID-* message with an attached `<UsageDirective>` header block MUST check the `actor` attribute and determine if it, the receiver, is the actor the header block is targeted at. If so, the receiver MUST check the `mustUnderstand` attribute. If set to `TRUE` the receiver MUST process the contents. If the attribute is absent or set to `FALSE` the receiver SHOULD attempt to process the content of the `<UsageDirective>` header block.

A receiver that processes the contents of a `<UsageDirective>` header block SHOULD verify the integrity of the header block – that is, it should verify any digital signatures that list the header block in its manifest [XMLDsig]. The receiver MUST verify that the `ref` attribute refers to an element in the message. That receiver MUST further process the message according to the policy expressed by the children elements of the `<UsageDirective>` header block. Those children elements will be imported from a foreign namespace, and MUST be parsed and interpreted according to the applicable schema and processing rules of that foreign namespace.

A receiver that cannot process a `<UsageDirective>` with `mustUnderstand` set to `TRUE` MUST respond with a `<S:Fault>`. The `<s:Fault>` MUST contain a `<detail>` element which in turn MUST contain a `<Status>` element with its `code` attribute set to `CannotHonourUsageDirective`. The `<Status>` element SHOULD possess a `ref` attribute with its value set to the value of the `id` attribute of the offending `<UsageDirective>` header block in the request message.

A receiver that cannot honor a non-mandatory (without `mustUnderstand` set to `TRUE`) `<UsageDirective>` must respond according to the contained policy. In addition, in this case the receiver MAY respond with a SOAP-based ID-* message that includes a `<Status>` element with its `code` attribute set to `CannotHonourUsageDirective`. This `<Status>` element instance SHOULD include a `ref` attribute with its value set to the value of the `id` attribute of the `<UsageDirective>` header block in the request message that could not be honored.

In this case, the receiver MAY include one or more new `<UsageDirective>` header blocks in its response message, each expressing a policy that the receiver would have been able to honor. The `ref` attribute of these headers SHOULD be set to the value of the `<wsa:MessageID>` header block in the request.

6.7. The `<ApplicationEPR>` Header Block

This section defines the `<ApplicationEPR>` header block. This header may be included in a message zero or more times and provides a means for a sender to specify application endpoints that may be referenced from the SOAP Body of the message.

The `<ApplicationEPR>` header block is an extension of `<wsa:EndpointReferenceType>`.

The schema fragment in Figure 12 defines the The `<ApplicationEPR>` header block.

```
<!-- application epr header block -->

<xs:element name="ApplicationEPR" type="wsa:EndpointReferenceType"/>
```

Figure 12. The <ApplicationEPR> Header Block Schema

```
<sb:ApplicationEPR S:mustUnderstand="1"
  S:actor="http://schemas.../next"
  wsu:Id="NotifyTo-001">
  <wsa:Address>...</wsa:Address>
</sb:ApplicationEPR>
```

Example 17. An instantiated <ApplicationEPR> header block

6.8. The <UserInteraction> Header Block

6.8.1. Overview

A WSC that interacts with a user (typically through a web-site offered by the WSC) may need to indicate its readiness to redirect the user agent of the user, or its readiness to pose questions to the user on behalf of other parties (such as WSPs). The <UserInteraction> header block provides a means by which a WSC can indicate its preferences and capabilities for interactions with requesting principals and, additionally, a SOAP fault message and HTTP redirect profile that enables the WSC and WSP to cooperate in redirecting the requesting principal to the WSP and, after browser interaction, back to the WSC.

6.8.2. UserInteraction Element

The <UserInteraction> element contains:

`InteractionService` [Optional]
: If present, this element MUST describe an interaction service hosted by the sender. This indicates that the sender can process messages defined for the interaction service [LibertyInteract], posing questions from the recipient of the message to the Principal.

`interact` [Optional]
: Indicates any preference that the sender has about interactions between the receiver and the requesting principal. The value is a string, for which we define the following values:

- *InteractIfNeeded* to indicate to the recipient that it should interact with the requesting principal if needed to satisfy the ID-WSF request. This is the default.
- *DoNotInteract* to indicate to the recipient that it MUST NOT interact with the requesting principal, either directly or indirectly. The sender prefers to receive an error response over the situation where the requesting principal would be distracted by an interaction.
- *DoNotInteractForData* to indicate to the recipient that it MAY interact with the requesting principal *only* if an explicit policy for the offered service so requires. The sender prefers to receive an error response over the situation where the WSP would obtain service response data (e.g., Personal Profile data) from the resource owner, but the sender does prefer to obtain a positive service response even if that requires policy-related interaction for, e.g., obtaining consent.

Note:

Implementors may choose to define additional values to indicate finer grained control over the user interactions.

`language` [Optional]
: This attribute indicates languages that the user is likely able to process. The value of this attribute is a space separated list of language identification tags ([RFC3066]). The WSC can obtain this information from the HTTP ([RFC2616]) *Accept-Language* header, or by other means, for example from a *personal profile service*.

`redirect` [Optional]
: An optional attribute to indicate that the sender supports the `<RedirectRequest>` element that a WSP may include in a message to the WSC. The value is *true* or *false*. When absent the default behavior will be as if *false*.

`maxInteractTime` [Optional]
: This is used to indicate the maximum time in seconds that the sender regards as reasonable for any possible interaction. The receiver is not expected to start any interaction if it has reason to assume that such an interaction is likely to take more time. In case an interaction is started and does seem to take longer the receiver is expected to respond with a message that contains a *InteractionTimeout* status code to the sender.

The schema fragment in Figure 13 defines the The `<UserInteraction>` header block.

```
<!-- user interaction header block -->

<xs:complexType name="UserInteractionHeaderType">
   <xs:sequence>
      <xs:element name="InteractionService" type="wsa:EndpointReferenceType"
         minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="interact" type="xs:string" use="optional" default="InteractIfNeeded"/>
   <xs:attribute name="language" type="xs:NMTOKENS" use="optional"/>
   <xs:attribute name="redirect" type="xs:boolean" use="optional" default="0"/>
   <xs:attribute name="maxInteractTime" type="xs:integer" use="optional"/>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="UserInteraction" type="UserInteractionHeaderType"/>
```

Figure 13. The `<UserInteraction>` Header Block Schema

6.8.3. UserInteraction Examples

Below is an example for a WSC that is prepared to redirect the user to a WSP, and also is ready to accept an `<is:InteractionRequest>`. The WSC wishes that the WSP will not attempt to prompt the resource owner for missing data; but accepts interactions for consent, as long as questioning the user will not take more than 60 seconds. The WSC expects the user to understand US English and Finnish.

```
<sb:UserInteraction interact="DoNotInteractForData" language="en-US fi"
            maxInteractTime="60" redirect="true">

  <sb:InteractionService xmlns:disco="urn:liberty:disco:2006-08">
    <wsa:Address>endpoint for interaction requests</wsa:Address>
    <wsa:Metadata>
      <disco:ServiceType>urn:liberty:is:2006-08</disco:ServiceType>
      <disco:Provider>http://someWSC</disco:Provider>
```

```
          <disco:Description>
            <disco:Endpoint>http://IS.com/soap</disco:Endpoint>
          </disco:Description>
        </wsa:Metadata>
      </sb:InteractionService>
    </sb:UserInteraction>
```

The following is an example for a WSC that wants to ensure that the WSP will not attempt to contact the requesting principal, even if this may hinder serving the actual request; the WSC would rather receive an error, or a less optimal response (e.g., fewer profile attributes).

```
        <sb:UserInteraction interact="DoNotInteract"/>
```

6.8.4. Processing Rules

If the sender includes an `InteractionService` element, it MUST set the value of `<disco:ServiceType>` within to *urn:liberty:is:2006-08*.

If the sender sets `interact="DoNotInteract"` it MUST omit the `InteractionService` element, as well as the `language`, `redirect` and `maxInteractTime` attributes.

The recipient of a message with a `UserInteraction` element MUST NOT respond with a `<RedirectRequest>` if the `redirect` is `false` or if `redirect` is absent.

The recipient MUST NOT start a requesting principal interaction if the `interact` attribute has a value of *"DoNotInteract"*.

The recipient MUST NOT interact with the requesting principal to obtain data that is to be included in a successful service response if the `interact` attribute has a value of *"DoNotInteractForData"*. In this case the recipient MAY start an interaction if a policy concerning available data so requires; for example if a policy requires that the Principal must be prompted for consent.

The recipient SHOULD NOT start a requesting principal interaction if it expects that the time to complete the interaction will exceed the value of the `maxInteractTime`.

The recipient MUST respond to the message after at most the number of seconds given as the value of the `maxInteractTime` attribute.

The sender must ensure that the `UserInteraction` element is integrity protected; i.e., if message level authentication (see [LibertySecMech]) is used the sender MUST sign the `UserInteraction` element. Likewise the receiver must ensure that the integrity of the `UserInteraction` element is not compromised, according to the processing rules in [LibertySecMech] .

6.8.4.1. UserInteraction Faults

A processor of a `UserInteraction` that must indicate an error situation related to this header SHOULD respond to the sender with an ID-WSF message that contains a `Status` element in the `detail` element of a `S:Fault`, or in a service specific `S:Body` component, or inside a higher level `Status` element. The `code` attribute of the included `Status` element can be set to one of the following values:

- *InteractionRequired*, as indication that the recipient has a need to start an interaction in order to satisfy the service request but the `interact` attribute value was set to *DoNotInteract*.

- *InteractionRequiredForData*. This indicates that the service request could not be satisfied because the WSP would have to interact with the requesting principal in order to obtain (some of) the requested data but the `interact` attribute value was set to *DoNotInteractForData*.

- *InteractionTimeNotSufficient*, as indication that the recipient has a need to start an interaction but has reason to believe that more time is needed that allowed for by the value of the `maxInteractTime` attribute.

- *InteractionTimeout*, as indication that the recipient could not satisfy the service request due to an unfinished interaction.

6.8.5. Cross-principal interactions

A 'cross-principal' interaction is defined by a WSC making a request on behalf of a principal who is different than the principal who 'owns' the resource in question. In such a case, the identity of the requesting principal will be identified by the security context of the message. The identity of the resource owner is expressed by the `<sb:TargetIndentity>` header.

Any `sb:UserInteraction` header in such a message always refers to the requesting principal. Consequently, if the WSP desires to interact with the requesting principal, it may use the interaction options as indicated by the `UserInteraction` (if present) or discover the requesting principal's permanent IS.

If the WSP desires to interact with the resource owner (as indicated by the `TargetIdentity` header), it will necessarily need to discover that principal's permanent IS as the alternative interaction mechanisms are not an option.

7. The `RedirectRequest` Protocol

In the `RedirectRequest` protocol the WSP requests the WSC to redirect the user agent of the Principal to a resource (URL) at the WSP. Once the user agent issues the HTTP request to fetch the URL the WSP has the opportunity to present one or more pages with questions and other information to the Principal. When the WSP has obtained the information that it required to serve the WSC, it redirects the user agent back to the WSC. The WSC can now re-issue its original request to the WSP. See [LibertyInteract] for an overview of various user interaction flows, including this redirect-based protocol.

7.1. `RedirectRequest` Element

The `RedirectRequest` element instructs the WSC to redirect the user to the WSP. It is an indication of the WSP that it cannot service a request made by the WSC before it obtains some more information from the user. The element is typically present in the `detail` element within a `<S:Fault>`. The `<RedirectRequest>` has one attribute:

`redirectURL` [Required] The URL to which the WSC should redirect the user agent. This URL MUST NOT contain parameters named `ReturnToURL` or `IDP` as these are reserved for the recipient of the `<RedirectRequest>` (see the `RedirectRequest` protocol). The URL SHOULD start with `https:` to ensure the establishment of a secure connection between the user agent and the WSP.

The optional text content of the element can be used to indicate the reason for the need for redirection of the requesting principal.

The schema fragment for the element is:

```
<xs:element name="RedirectRequest" type="RedirectRequestType"/>
<xs:complexType name="RedirectRequestType">
  <xs:attribute name="redirectURL" type="xs:anyURI" use="required"/>
</xs:complexType>
```

An example of a `<RedirectRequest>` in a SOAP Fault could look like:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/02/addressing">...</wsa:MessageID>
    <wsa:RelatesTo>...</wsa:RelatesTo>
  </S:Header>
  <S:Body>
    <S:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>
        <RedirectRequest redirectURL="https://someWSP/getConsent?transID=de67hj89jk65nk34">
          Redirecting to AP to obtain consent
        </RedirectRequest>
      </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

7.1.1. Processing Rules

The recipient of a `<RedirectRequest>` MUST verify that the `redirectURL` points to the WSP, i.e., the host in the URL should be the same as the host to which the WSC sent its service request. If this is not the case the recipient MUST ignore the `<RedirectRequest>`.

The recipient MUST attempt to direct the user agent to issue an HTTP request ([RFC2616]) for the URL in the `redirectURL` attribute of the `<RedirectRequest>`. That user agent MUST be associated with the ID-WSF request that caused the `<RedirectRequest>`. The recipient MUST add a `ReturnToURL` parameter to the `redirectURL` with its value the URL-encoded URL which the recipient wants the user agent directed back to. It is recommended that this `ReturnToURL` includes an identifier that associates the URL to the originating ID-WSF message to the WSP. The recipient MAY add an `IDP` parameter to the `redirectURL` with its value the `providerID` of an identity provider that was used to authenticate the user to the WSC.

The recipient may instruct the user agent to submit either an HTTP GET or an HTTP POST request to the URL; in this way the WSC can avoid problems with user agents that can handle only short URLs. If the user agent is instructed to submit a HTTP POST, *all* URL parameters should be form-encoded, and the HTTP content-type header of the request MUST be `application/x-www-form-urlencoded`. Note that this implies that the WSP SHOULD accept both an HTTP GET as well as an HTTP POST request for the `redirectURL`, but in either case retrieval of URL parameters can be done using well-known techniques; most HTTP server environments effectively encapsulate the different methods for submission of parameters.

As an example, assume that a Principal visits a service provider. As a result the service provider (acting as WSC) could have made a request to a WSP, and that WSP would have responded with a SOAP Fault similar to that of the example above. The WSC would now send a HTTP response to the user agent that would look like:

```
HTTP 302
Location:
https://someWSP/getConsent?transID=de67hj89jk65nk34&ReturnToURL=
https%3a%2f%2fsomeWSC%2fisReturn3bjsession%3d9A6F2E3A&IDP=1A2B3C4D5E1A2B3C4D5E
... other HTTP headers ...

<html>
   <head>
      <title>Redirecting...</title>
   </head>
   <body>
      <p>Redirecting to AP to obtain consent</p>
   </body>
</html>
```

The WSC appends its own `ReturnToURL` as a parameter to the value of the `redirectURL` element that the WSC specified in its `RedirectRequest`.

7.2. `RedirectRequest` Protocol

The `<RedirectRequest>` protocol consists of the following steps, each with normative rules:

7.2.1. Step 1: WSC Issues Normal ID-WSF Request

For the `<RedirectRequest>` protocol to be initiated the originating ID-WSF message MUST contain a `UserInteraction` element with its `redirect` attribute set to *true*.

The ID-WSF message SHOULD contain a `wsa:FaultTo` element to which the WSC desires fault messages be sent.

7.2.2. Step 2: WSP Responds with `<RedirectRequest>`

If, and only if, an ID-WSF message contains a `<UserInteraction>` element with its `redirect` attribute set to *true* MAY the recipient of the ID-WSF message respond with a `<RedirectRequest>` message in a SOAP Fault.

Note:

The `redirectURL` attribute MUST be constructed as to include the necessary information for mapping the upcoming HTTP request to the originating ID-WSF message; for example by inclusion of the value of the `wsa:MessageID` Header from that message.

7.2.3. Step 3: WSC Instructs User Agent to Contact the WSP

When the WSC receives a `<RedirectRequest>` it MUST attempt to direct the user agent to issue an HTTP request for the URL in the `redirectURL` attribute of the `RedirectRequest`. The user agent MUST be associated with the ID-WSF message that caused the `<RedirectRequest>`. The WSC MUST append a `ReturnToURL` parameter to the `redirectURL` with its value the URL-encoded URL to which the WSC wants the user agent directed back.

Note:

How this step is performed will depend on the user agent. In most cases it is accomplished by a simple HTTP 302 response with a `Location` header set to the `redirectURL`. Different user agents may be better served by other approaches, for example a WML browser may be able to handle a redirect deck better than a potentially long URL. See the processing rules for the `<RedirectRequest>`.

7.2.4. Step 4: WSP Interacts with User Agent

In step 4 the user agent issues the HTTP request for the `redirectURL`, with the `ReturnToURL` parameter appended, with any `IDP` parameter also appended. The WSP MUST verify that the `ReturnToURL` points to the WSC, i.e., the host in the URL should be the same as the host to which the WSP sent the `<RedirectRequest>`. If this is not the case the WSP MUST ignore the `ReturnToURL`, abort the protocol, and construct a meaningful error message for the user. If verification succeeds, however, the service (WSP) MAY now proceed with a HTTP response that contains an inquiry directed at the user. The WSP SHOULD verify that the identity of the user is that of the owner of the resource that was targeted in the originating ID-WSF request, for example by means of a `<saml:AuthnRequest>` (see [SAMLCore2]). This step may be followed by any number of interactions between the user and the WSP, but the WSP should attempt to execute step 5 within a reasonable time.

7.2.5. Step 5: WSP Redirects User Agent Back to WSC

In step 5 the WSP that issued the `<RedirectRequest>` MUST attempt to instruct the user agent to issue an HTTP request for the `ReturnToURL` that was included as parameter on the URL of the HTTP request made in step 4. The WSP SHOULD append a `ResendMessage` parameter to the `ReturnToURL`. This parameter serves as a hint to the WSC about the next step. A value of *0* or *false* indicates that the WSC should not try to re-issue the originating ID-WSF request, presumably because the resource owner did not approve completion of the transaction. If the value of `ResendMessage` is *true*, *1*, or any string other than *0* or *false*, it is an indication that the WSP recommends that the WSC re-issue the originating request. It is RECOMMENDED that in this situation, the value of this parameter be set to the value of the `wsa:MessageID` element of the originating ID-WSF message.

7.2.6. Step 6: User Agent Requests `ReturnToURL` from WSC

In step 6 the user agent requests the `ReturnToURL` from the WSC. The WSC SHOULD check the value of the `ResendMessage` parameter; if the value is *0* or *false* the WSC SHOULD NOT send an ID-WSF message with a request for the same resource and/or action (as in step 1). If the value of the `ResendMessage` parameter is anything else, then the WSC MAY resend the message (Step 7).

After receiving the response from the WSP, the WSC should send a HTTP response to the user agent.

7.2.7. Step 7: WSC Resends Message

If the WSC resends its request it MUST set the value of the `wsa:RelatesTo` SOAP Header to the same value of the `wsa:MessageID` SOAP Header of the SOAP Fault that carried the `<RedirectRequest>` element (in step 2). .

7.2.8. Steps 8: WSP sends response

The WSP responds to the WSC's second request. The WSP MUST set the value of the `wsa:RelatesTo` SOAP Header to the same value of the `wsa:MessageID` SOAP Header of the WSC's resent request.

7.2.9. Steps 9: WSC sends HTTP response to User Agent

Finally, the WSC returns an HTTP response to the user agent.

8. Security Considerations

- The header blocks specified in this document should be integrity-protected using the mechanisms detailed in [LibertySecMech].
- Header blocks should be signed in accordance with [LibertySecMech]. The receiver of a message containing a signature that covers specific header blocks should verify the signature as part of verifying the integrity of the header block.
- Metadata [LibertyMetadata] should be used to the greatest extent possible to verify message sender identity claims.
- Message senders and receivers should be authenticated to one another via the mechanisms discussed in [Liberty-SecMech].
- To prevent message replay, receivers should maintain a message cache, and check received `messageID` values against the cache.

9. Acknowledgements

The members of the Liberty Technical Expert group, especially Greg Whitehead, Jonathan Sergent, Xavier Serret, and Conor Cahill, provided valuable input to this specification. The docbook source code for this specification was hand set to the tunes of The Sugarcubes, King Crimson, Juliana Hatfield, Smashing Pumpkins, Evanescence, Mad at Gravity, Elisa Korenne, The Breeders, fIREHOSE, Polly Jean Harvey, Jimi Hendrix, and various others.

References

Normative

[LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02, Liberty Alliance Project (25 November 2004). *http://www.projectliberty.org/specs*

[LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[RFC2045] Freed, N., Borenstein, N., eds. (November 1996). "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies ," RFC 2045., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2045.txt*

[RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet Engineering Task Force (March 1997). *http://www.ietf.org/rfc/rfc2119.txt*

[RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2828.txt*

[RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also STD0066) (Status: STANDARD), The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3986.txt*

[SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf*

[SAMLGloss2] Hodges, Jeff, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML 2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf*

[Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-1/*

[Schema2-2] Biron, Paul V., Malhotra, Ashok, eds. (28 October 2004). "XML Schema Part 2: Datatypes Second Edition," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-2/*

[SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C Note (08 May 2000). *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*

[SOAPv1.1-Schema] "SOAP 1.1 Envelope schema," W3C W3C Note *http://schemas.xmlsoap.org/soap/envelope/*

[WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith, Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001). *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004). "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/2004/REC-xml-20040204*

[XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and Processing," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmldsig-core*

[WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). *http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/*

[WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). *http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/*

[WSAv1.0-Schema] "WS-Addressing 1.0 Schema," W3C, W3C Working Draft *http://dev.w3.org/cvsweb/~checkout~/2004/ws/addressin addr.xsd*

[wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf*

Informational

[LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (29 November, 2006). *http://www.projectliberty.org/specs*

[LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyIDWSF20SCR] Whitehead, Greg, eds. Version 1.0 errata v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification," Version 1.1, Liberty Alliance Project (29 September, 2005). *http://www.projectliberty.org/specs*

[LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata," Version 1.0, Liberty Alliance Project (13 April, 2007). *http://www.projectliberty.org/specs*

[Merriam-Webster] "Merriam-Webster Dictionary," *http://www.merriam-webster.com/*

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2616.txt*

[RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3066.txt*

[RFC4086] Eastlake , D., Schiller, J., Crocker, S., eds. (June 2005). "Randomness Recommendations for Security ," RFC 4086, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc4086.txt*

[SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah, Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Recommendation (07 May 2003). *http://www.w3.org/TR/2003/PR-soap12-part1-20030507/*

A. liberty-idwsf-soap-binding.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:sb"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:liberty:sb"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

   <!-- framework header block -->

   <xs:complexType name="FrameworkType">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:element name="Framework" type="FrameworkType"/>

</xs:schema>
```

B. liberty-idwsf-soap-binding-v2.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:sb:2006-08"
   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns="urn:liberty:sb:2006-08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

   <xs:import
      namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecur
ity-utility-1.0.xsd"
      schemaLocation="wss-util-1.0.xsd"/>

   <xs:import
      namespace="urn:oasis:names:tc:SAML:2.0:protocol"
      schemaLocation="saml-schema-protocol-2.0.xsd"/>

   <xs:import
      namespace="http://www.w3.org/2005/08/addressing"
      schemaLocation="ws-addr-1.0.xsd"/>

   <xs:import
      namespace="urn:liberty:util:2006-08"
      schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>

   <!-- sender header block -->

   <xs:complexType name="SenderType">
      <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
      <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>

    <xs:element name="Sender" type="SenderType"/>


   <!-- target identity header block -->

   <xs:complexType name="TargetIdentityType">
      <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:element name="TargetIdentity" type="TargetIdentityType"/>


   <!-- credentials context header block -->

   <xs:complexType name="CredentialsContextType">
      <xs:sequence>
         <xs:element ref="samlp:RequestedAuthnContext" minOccurs="0"/>
         <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:element name="CredentialsContext" type="CredentialsContextType"/>


   <!-- epr update header block -->
```

```
  <xs:complexType name="EndpointUpdateType">
     <xs:complexContent>
        <xs:extension base="wsa:EndpointReferenceType">
           <xs:attribute name="updateType" type="xs:anyURI" use="optional"/>
        </xs:extension>
     </xs:complexContent>
  </xs:complexType>

  <xs:element name="EndpointUpdate" type="EndpointUpdateType"/>


  <!-- timeout header block -->

  <xs:complexType name="TimeoutType">
     <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
     <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:element name="Timeout" type="TimeoutType"/>


<!-- processing context header block -->

<xs:complexType name="ProcessingContextType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="ProcessingContext" type="ProcessingContextType"/>

<!-- consent header block -->

<xs:complexType name="ConsentType">
  <xs:attribute name="uri" type="xs:anyURI" use="required"/>
  <xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="Consent" type="ConsentType"/>

<!-- usage directive header block -->

<xs:complexType name="UsageDirectiveType">
  <xs:sequence>
    <xs:any namespace="##other" processContents="lax"
      maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:IDREF" use="required"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="UsageDirective" type="UsageDirectiveType"/>

  <!-- application epr header block -->

  <xs:element name="ApplicationEPR" type="wsa:EndpointReferenceType"/>

<!-- user interaction header block -->

<xs:complexType name="UserInteractionHeaderType">
   <xs:sequence>
      <xs:element name="InteractionService" type="wsa:EndpointReferenceType"
         minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xs:sequence>
    <xs:attribute name="interact" type="xs:string" use="optional" default="InteractIfNeeded"/>
    <xs:attribute name="language" type="xs:NMTOKENS" use="optional"/>
    <xs:attribute name="redirect" type="xs:boolean" use="optional" default="0"/>
    <xs:attribute name="maxInteractTime" type="xs:integer" use="optional"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:element name="UserInteraction" type="UserInteractionHeaderType"/>
   <xs:element name="RedirectRequest" type="RedirectRequestType"/>
   <xs:complexType name="RedirectRequestType">
    <xs:attribute name="redirectURL" type="xs:anyURI" use="required"/>
   </xs:complexType>
</xs:schema>
```

C. liberty-idwsf-utility-v2.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:liberty:util:2006-08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="2.0-03">

   <xs:annotation>
      <xs:documentation>
   Liberty Alliance Project utility schema.  A collection of common
   IDentity Web Services Framework (ID-WSF) elements and types.
   This schema is intended for use in ID-WSF schemas.

   This version: 2006-08

         Copyright (c) 2006 Liberty Alliance participants, see
         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
      </xs:documentation>
   </xs:annotation>
   <xs:simpleType name="IDType">
      <xs:annotation>
         <xs:documentation>
            This type should be used to provide IDs to components
            that have IDs that may not be scoped within the local
            xml instance document.
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:simpleType name="IDReferenceType">
      <xs:annotation>
         <xs:documentation>
            This type can be used when referring to elements that are
            identified using an IDType.
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:attribute name="itemID" type="IDType"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   <xs:complexType name="StatusType">
      <xs:annotation>
         <xs:documentation>
            A type that may be used for status codes.
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="code" type="xs:string" use="required"/>
      <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
      <xs:attribute name="comment" type="xs:string" use="optional"/>
   </xs:complexType>

   <xs:element name="Status" type="StatusType">
      <xs:annotation>
         <xs:documentation>
            A standard Status type
         </xs:documentation>
      </xs:annotation>
   </xs:element>

   <xs:complexType name="ResponseType">
      <xs:sequence>
```

```
         <xs:element ref="Status"     minOccurs="1" maxOccurs="1"/>
         <xs:element ref="Extension"   minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="itemIDRef" use="optional"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
   <xs:element name="TestResult" type="TestResultType"/>
   <xs:complexType name="TestResultType">
      <xs:simpleContent>
         <xs:extension base="xs:boolean">
            <xs:attribute ref="itemIDRef" use="required"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="EmptyType">
      <xs:annotation>
         <xs:documentation> This type may be used to create an empty element </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
         <xs:restriction base="xs:anyType"/>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="Extension" type="extensionType">
      <xs:annotation>
         <xs:documentation>
            An element that contains arbitrary content extensions
            from other namespaces
         </xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="extensionType">
      <xs:annotation>
         <xs:documentation>
            A type for arbitrary content extensions from other namespaces
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

D. liberty-utility-v2.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="2.0-01">
   <xs:annotation>
      <xs:documentation>
   Liberty Alliance Project utility schema.  A collection of common
   elements and types for use with independent Liberty XML Schema documents.

   This file intended for inclusion, rather than importation, into other schemas.
   This version: 2004-12

         Copyright (c) 2004 Liberty Alliance participants, see
         http://www.projectliberty.org/specs/idff_copyrights.html
      </xs:documentation>
   </xs:annotation>
   <xs:simpleType name="IDType">
      <xs:annotation>
         <xs:documentation>
            This type should be used to provide IDs to components
            that have IDs that may not be scoped within the local
            xml instance document.
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:simpleType name="IDReferenceType">
      <xs:annotation>
         <xs:documentation>
            This type can be used when referring to elements that are
            identified using an IDType.
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:complexType name="StatusType">
      <xs:annotation>
         <xs:documentation>
            A type that may be used for status codes.
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="code" type="xs:string" use="required"/>
      <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
      <xs:attribute name="comment" type="xs:string" use="optional"/>
   </xs:complexType>

   <xs:element name="Status" type="StatusType">
      <xs:annotation>
         <xs:documentation>
            A standard Status type
         </xs:documentation>
      </xs:annotation>
   </xs:element>

   <xs:complexType name="EmptyType">
      <xs:annotation>
         <xs:documentation> This type may be used to create an empty element </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
         <xs:restriction base="xs:anyType"/>
      </xs:complexContent>
```

```
   </xs:complexType>
   <xs:element name="Extension" type="extensionType">
      <xs:annotation>
         <xs:documentation>
            An element that contains arbitrary content extensions
            from other namespaces
         </xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="extensionType">
      <xs:annotation>
         <xs:documentation>
            A type for arbitrary content extensions from other namespaces
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

E. wss-util-1.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
OASIS takes no position regarding the validity or scope of any intellectual property or other
rights that might be claimed to pertain to the implementation or use of the technology described
in this document or the extent to which any license under such rights might or might not be
available; neither does it represent that it has made any effort to identify any such rights.
Information on OASIS's procedures with respect to rights in OASIS specifications can be found
at the OASIS website. Copies of claims of rights made available for publication and any
assurances of licenses to be made available, or the result of an attempt made to obtain a
general license or permission for the use of such proprietary rights by implementors or users
of this specification, can be obtained from the OASIS Executive Director.
OASIS invites any interested party to bring to its attention any copyrights, patents or
patent applications, or other proprietary rights which may cover technology that may be
required to implement this specification. Please address the information to the OASIS Executive Director.

Copyright © OASIS Open 2002-2004. 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 implementation 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 does not be modified in any way, such
as by removing the copyright notice or references to OASIS, except as needed for the purpose
of developing OASIS specifications, in which case the procedures for copyrights defined
in the OASIS Intellectual Property Rights document must be followed, or as required to
translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS
or its successors or assigns.
This document and the information contained herein is provided on an "AS IS" basis
and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
-->
<xsd:schema targetNamespace="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"



xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.
0.xsd" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
   <!-- // Fault Codes ////////////////////////////////////////// -->
   <xsd:simpleType name="tTimestampFault">
      <xsd:annotation>
         <xsd:documentation>
This type defines the fault code value for Timestamp message expiration.
       </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:QName">
         <xsd:enumeration value="wsu:MessageExpired"/>
      </xsd:restriction>
   </xsd:simpleType>
   <!-- // Global attributes ////////////////////////////////// -->
   <xsd:attribute name="Id" type="xsd:ID">
      <xsd:annotation>
         <xsd:documentation>
This global attribute supports annotating arbitrary elements with an ID.
       </xsd:documentation>
      </xsd:annotation>
   </xsd:attribute>
   <xsd:attributeGroup name="commonAtts">
      <xsd:annotation>
         <xsd:documentation>
Convenience attribute group used to simplify this schema.
       </xsd:documentation>
      </xsd:annotation>
```

```
      <xsd:attribute ref="wsu:Id" use="optional"/>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
   </xsd:attributeGroup>
   <!-- // Utility types ////////////////////////////////////// -->
   <xsd:complexType name="AttributedDateTime">
      <xsd:annotation>
         <xsd:documentation>
This type is for elements whose [children] is a psuedo-dateTime and can have arbitrary attributes.
    </xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
         <xsd:extension base="xsd:string">
            <xsd:attributeGroup ref="wsu:commonAtts"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="AttributedURI">
      <xsd:annotation>
         <xsd:documentation>
This type is for elements whose [children] is an anyURI and can have arbitrary attributes.
    </xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
         <xsd:extension base="xsd:anyURI">
            <xsd:attributeGroup ref="wsu:commonAtts"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <!-- // Timestamp header components ///////////////////////////// -->
   <xsd:complexType name="TimestampType">
      <xsd:annotation>
         <xsd:documentation>
This complex type ties together the timestamp related elements into a composite type.
         </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element ref="wsu:Created" minOccurs="0"/>
         <xsd:element ref="wsu:Expires" minOccurs="0"/>
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:any namespace="##other" processContents="lax"/>
         </xsd:choice>
      </xsd:sequence>
      <xsd:attributeGroup ref="wsu:commonAtts"/>
   </xsd:complexType>
   <xsd:element name="Timestamp" type="wsu:TimestampType">
      <xsd:annotation>
         <xsd:documentation>
This element allows Timestamps to be applied anywhere element wildcards are present,
including as a SOAP header.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <!-- global element decls to allow individual elements to appear anywhere -->
   <xsd:element name="Expires" type="wsu:AttributedDateTime">
      <xsd:annotation>
         <xsd:documentation>
This element allows an expiration time to be applied anywhere element wildcards are present.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
   <xsd:element name="Created" type="wsu:AttributedDateTime">
      <xsd:annotation>
         <xsd:documentation>
This element allows a creation time to be applied anywhere element wildcards are present.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
```

```
</xsd:schema>
```

F. ws-addr-1.0.xsd Schema Listing

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN" "http://www.w3.org/2001/XMLSchema.dtd">

<!--
   W3C XML Schema defined in the Web Services Addressing 1.0 specification
   http://www.w3.org/TR/ws-addr-core

  Copyright © 2005 World Wide Web Consortium,

  (Massachusetts Institute of Technology, European Research Consortium for
  Informatics and Mathematics, Keio University). All Rights Reserved. This
  work is distributed under the W3C® Software License [1] in the hope that
  it will be useful, but WITHOUT ANY WARRANTY; without even the implied
  warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

  [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

  $Id: ws-addr-1.0.xsd 3060 2005-09-23 18:20:29Z dchampagne $
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://www.w3.org/2005/08/addressing"
        targetNamespace="http://www.w3.org/2005/08/addressing"
        blockDefault="#all"
        elementFormDefault="qualified"
        finalDefault=""
        attributeFormDefault="unqualified">

   <!-- Constructs from the WS-Addressing Core -->

   <xs:element name="EndpointReference" type="tns:EndpointReferenceType"/>
   <xs:complexType name="EndpointReferenceType" mixed="false">
      <xs:sequence>
         <xs:element name="Address" type="tns:AttributedURIType"/>
         <xs:element name="ReferenceParameters" type="tns:ReferenceParametersType" minOccurs="0"/>
         <xs:element ref="tns:Metadata" minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:complexType name="ReferenceParametersType" mixed="false">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:element name="Metadata" type="tns:MetadataType"/>
   <xs:complexType name="MetadataType" mixed="false">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:element name="MessageID" type="tns:AttributedURIType"/>
   <xs:element name="RelatesTo" type="tns:RelatesToType"/>
   <xs:complexType name="RelatesToType" mixed="false">
      <xs:simpleContent>
         <xs:extension base="xs:anyURI">
            <xs:attribute name="RelationshipType" type="tns:RelationshipTypeOpenEnum" use="optional"

               default="http://www.w3.org/2005/08/addressing/reply"/>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
```

```
      </xs:simpleContent>
   </xs:complexType>

   <xs:simpleType name="RelationshipTypeOpenEnum">
      <xs:union memberTypes="tns:RelationshipType xs:anyURI"/>
   </xs:simpleType>

   <xs:simpleType name="RelationshipType">
      <xs:restriction base="xs:anyURI">
         <xs:enumeration value="http://www.w3.org/2005/08/addressing/reply"/>
      </xs:restriction>
   </xs:simpleType>

   <xs:element name="ReplyTo" type="tns:EndpointReferenceType"/>
   <xs:element name="From" type="tns:EndpointReferenceType"/>
   <xs:element name="FaultTo" type="tns:EndpointReferenceType"/>
   <xs:element name="To" type="tns:AttributedURIType"/>
   <xs:element name="Action" type="tns:AttributedURIType"/>

   <xs:complexType name="AttributedURIType" mixed="false">
      <xs:simpleContent>
         <xs:extension base="xs:anyURI">
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <!-- Constructs from the WS-Addressing SOAP binding -->

   <xs:attribute name="IsReferenceParameter" type="xs:boolean"/>

   <xs:simpleType name="FaultCodesOpenEnumType">
      <xs:union memberTypes="tns:FaultCodesType xs:QName"/>
   </xs:simpleType>

   <xs:simpleType name="FaultCodesType">
      <xs:restriction base="xs:QName">
         <xs:enumeration value="tns:InvalidAddressingHeader"/>
         <xs:enumeration value="tns:InvalidAddress"/>
         <xs:enumeration value="tns:InvalidEPR"/>
         <xs:enumeration value="tns:InvalidCardinality"/>
         <xs:enumeration value="tns:MissingAddressInEPR"/>
         <xs:enumeration value="tns:DuplicateMessageID"/>
         <xs:enumeration value="tns:ActionMismatch"/>
         <xs:enumeration value="tns:MessageAddressingHeaderRequired"/>
         <xs:enumeration value="tns:DestinationUnreachable"/>
         <xs:enumeration value="tns:ActionNotSupported"/>
         <xs:enumeration value="tns:EndpointUnavailable"/>
      </xs:restriction>
   </xs:simpleType>

   <xs:element name="RetryAfter" type="tns:AttributedUnsignedLongType"/>
   <xs:complexType name="AttributedUnsignedLongType" mixed="false">
      <xs:simpleContent>
         <xs:extension base="xs:unsignedLong">
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

   <xs:element name="ProblemHeaderQName" type="tns:AttributedQNameType"/>
   <xs:complexType name="AttributedQNameType" mixed="false">
      <xs:simpleContent>
         <xs:extension base="xs:QName">
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
      </xs:simpleContent>
```

```
   </xs:complexType>

   <xs:element name="ProblemHeader" type="tns:AttributedAnyType"/>
   <xs:complexType name="AttributedAnyType" mixed="false">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

   <xs:element name="ProblemIRI" type="tns:AttributedURIType"/>

   <xs:element name="ProblemAction" type="tns:ProblemActionType"/>
   <xs:complexType name="ProblemActionType" mixed="false">
      <xs:sequence>
         <xs:element ref="tns:Action" minOccurs="0"/>
         <xs:element name="SoapAction" minOccurs="0" type="xs:anyURI"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>

</xs:schema>
```