Liberty ID-WSF SOAP Binding Specification

Version: 1.2

Editors:
Jeff Hodges, Sun Microsystems, Inc.
John Kemp, Nokia Corporation
Robert Aarts, Nokia Corporation

Contributors:
Conor Cahill, AOL Time Warner, Inc.
Marc Hadley, Sun Microsystems, Inc.
Jukka Kainulainen, Nokia Corporation
Jonathan Sergent, Sun Microsystems, Inc.
Greg Whitehead, Trustgenix, 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 simple message correlation, as well as provider declaration, processing context, consent claims, usage directives and a number of other optional headers.

Filename: liberty-idwsf-soap-binding-v1.2.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.** Implementors 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

Liberty Identity Web Services Framework (ID-WSF) and Service Interface Specification (ID-SIS) messages [LibertyIDWSFOverview], collectively referred to as *ID-* messages* in this specification, are designed so that they 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].

This specification defines a number of SOAP header blocks, addressing specific aspects of ID-* message exchange functionality. They are:

- Message Correlation:
  SOAP does not define a mechanism to correlate one SOAP message with another message, such as in a request-response paradigm. This specification defines the `<Correlation>` header block for this purpose.

- Provider and affiliation declaration:
  Participants in ID-* interactions may declare themselves by their Provider ID, as well as their Affiliation ID if appropriate. This specification defines the `<Provider>` header block for this purpose.

- 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.

- 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.

- 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.

- 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.

- Service Instance Update:
  The `<ServiceInstanceUpdate>` 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.

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] and [LibertyInteract], 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] 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.

Table 1. XML Namespaces and Prefixes

Prefix	Namespace
sb:	Represents the Liberty SOAP Binding namespace: **`urn:liberty:sb:2003-08`** **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.
sb-ext:	Represents a Liberty SOAP Binding Extensions namespace: **`urn:liberty:sb:2004-04`**
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].
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 [LibertySecMech].
xs:	Represents the W3C XML schema namespace: `http://www.w3.org/2001/XMLSchema` This namespace is defined in [Schema1].

2.2. Terminology

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

affiliation
: An *affiliation* is a set of one or more entities, described by *Provider IDs*, who may perform Liberty interactions as a member of the set. An affiliation is referenced by exactly one *Affiliation ID*, and is administered by exactly one entity identified by their Provider ID. 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* who 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
: An *ID-* fault message* is a SOAP `<S:Fault>` element containing a `<Status>` element, with the attributes and attribute values of both elements configured as specified herein, or as specified in other specification(s) in the ID-WSF or ID-SIS specification sets.

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 a synchronous 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 `<Correlation>` header block `refToMessageID` attribute value being set, by the sender, from the previous successfully received message's `<Correlation>` header block `messageID` attribute value.

ordinary ID-* message
: An *ordinary ID-* message* is a Liberty Identity Web Services Framework (ID-WSF) or Service Interface Specification (ID-SIS) message as defined in the [LibertyDST], [LibertyDisco], and [LibertyIDPP] specifications and others. It has the characteristics of being designed to be conveyed by essentially any transport or transfer protocol, notably SOAP [SOAPv1.1]. It is also known among the ID-* specifications as a *service request*, or an ID-WSF (service) request, or an ID-SIS (service) request.

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
: A *SOAP message* conveying ID-WSF and perhaps ID-SIS header blocks and conveying either an *ordinary ID-* message* or an *ID-* fault message*. After being bound to SOAP, the resultant composite messages are referred to as an *Ordinary SOAP-bound ID-* Message* and a *SOAP-bound ID-* Fault Message*, respectively.

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 [SAMLGloss].

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 [RFC2396] 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]. 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:

The ID types described in Section 3.2: "ID" Types and Message Identifiers are `string`-based.

Also, various element and/or attribute components of the schema described by this specification (see Appendix A: SOAP Binding Schema , below) may have further requirements placed on the values they may take on. For example, see Section 5.1.2: `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] 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] files ("Liberty ID-WSF SOAP Binding Schema v1.0, v1.1", reproduced in Appendix A, Appendix B). In addition, the Liberty ID-WSF SOAP Binding Schema files explicitly include, 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 SOAPv1.1 schema [SOAPv1.1-Schema] (reproduced in Appendix D).

3.2. "ID" Types and Message Identifiers

The `IDType` simple type is used in this specification to declare *message identifiers*. The `IDReferenceType` is used to reference message identifiers.

The schema fragment in Figure 1, from the ID-WSF Utility schema (Appendix C), defines the `IDType` and `IDReferenceType` simple types.

```
   <xs:simpleType name="IDType">
      <xs:annotation>
         <xs:documentation>This type should be used to provided 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>
```

Figure 1. IDType and IDReferenceType Schema

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 2, from the ID-WSF Utility schema (Appendix C), shows both the `<Status>` element and `StatusType` complex type.

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

<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:QName" use="required"/>
   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
   <xs:attribute name="comment" type="xs:string" use="optional"/>
</xs:complexType>
```

Figure 2. `Status` and `StatusType` Schema

3.3.1. Status Codes

This section lists, in Table 2, the values defined in this specification for the `code` attribute of the `<Status>` element.

282 **Table 2. Status Codes**

Code	Semantics	Suggested Fault Source
sb:InvalidActor	There is an issue with the `actor` attribute on the indicated header block in the indicated message.	S:Client
sb:InvalidMustUnderstand	There is an issue with the `mustUnderstand` attribute on the indicated header block in the indicated message.	S:Client
sb:StaleMsg	The indicated inbound SOAP-bound ID-* message has a correlation timestamp value outside of the receivers allowable time window.	S:Client
sb:DuplicateMsg	The indicated inbound SOAP-bound ID-* message appears to be a duplicate.	S:Client
sb:InvalidRefToMsgID	The indicated inbound SOAP-bound ID-* message appears to incorrectly refer to the preceding message in the message thread.	S:Client
sb:ProviderIDNotValid	The receiver does not consider the claimed Provider ID to be valid.	S:Client
sb:AffiliationIDNotValid	The receiver does not consider the claimed Affiliation ID to be valid.	S:Server
sb:IDStarMsgNotUnderstood	There was a problem with understanding/parsing the conveyed ID-* message.	S:Client
sb:ProcCtxURINotUnderstood	The receiver did not understand the processing context facet URI.	S:Server
sb:ProcCtxUnwilling	The receiver is unwilling to apply the sender's stipulated processing context.	S:Server
sb:CannotHonourUsageDirective	The receiver is unable or unwilling to honor the stipulated usage directive.	S:Server
sb-ext:EndpointMoved	The request cannot be processed at this endpoint. This is typically used in conjunction with the `<ServiceInstanceUpdate>` header block to indicate the endpoint to which the request should be re-submitted.	S:Server
sb-ext: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 `<ServiceInstanceUpdate>` header block.	S:Client
sb-ext: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

283 3.4. SOAP Fault Types

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

285 The schema fragment in Figure 3, extracted from [SOAPv1.1-Schema], defines the SOAPv1.1 `Fault` and `detail`
286 complex types, which define the `<S:Fault>` and `<detail>` elements, respectively.

287 **Note:**

288 The `<S:Fault>` element is **not** intended to be used as a SOAP header block. Rather, it is designed to be
289 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 3. 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 5.3: 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. When a SOAP message conformant to this specification is emitted bound to HTTP, according to the binding specified in [SOAPv1.1], the SOAPAction HTTP header SHOULD be set to "" (an empty string).

Note:

It should be noted 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. A tool that generates message-producing code might thus auto-generate code that adds a SOAPAction header of a different value than that specified above.

4.3. Ordinary ID-* Messages

Ordinary ID- Messages* are messages of the forms defined in Liberty ID-WSF and ID-SIS specification sets (see Section 2.2: Terminology, above). For example, these are the class of messages defined in [LibertyDST], or the Discovery Service specification [LibertyDisco], or in the ID-PersonalProfile specification [LibertyIDPP]. See Example 1. These messages must 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) containing a <Status> element, with the attributes and attribute values of both elements configured as specified herein, or as specified in other specification(s) in the ID-WSF or ID-SIS specification sets.

The <S:Fault> element's attributes and child elements MUST be tailored according to these rules:

1. The <S:Fault> element:

A. SHOULD contain a `<faultcode>` element whose value SHOULD be either "`S:server`" or "`S:client`".
Note:
A `<faultcode>` of "`S:server`" indicates that the receiver believes that it has erred, whereas "`S:client`" is intended for cases in which the receiver believes that the sender has erred. Such an indication should be considered merely informational.

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 MUST 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.

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. The `<S:Fault>` element's `<detail>` child element SHOULD also contain an indicator of which header block or ID-* message body element, in the message being processed, is being referred to by the fault. The indicator is constructed by including in the `<detail>` element, after the `<Status>` element, an element of the same type as the one referred to by the fault. This element SHOULD have only one attribute, the `id` attribute whose value MUST be the value of the `id` attribute of the element referred to by the fault.

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 a `<Correlation>` header block, and possibly a `<Provider>` header block in its `<S:Header>` element, and,
- perhaps having other Liberty ID-WSF or ID-SIS 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 5.3: 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:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    <!-- other header blocks, eg wsse:security, may go here -->

    <Correlation S:mustUnderstand="1"
      id="A13454...245"
      actor="http://schemas.../next"
      messageID="uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
      timestamp="2112-03-15T11:12:12Z"/>

    <Provider providerID="example.com"
      affiliationID="affiliation.example.com"
      S:mustUnderstand="1"
      id="A9kendan...542"
      actor="http://schemas.../next"/>

    <!-- other header blocks, eg wsse:security, may go here -->

  </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="urn:liberty:sb:2003-08"
  xmlns:pp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    <!-- other header blocks, eg wsse:security, may go here -->

    <Correlation S:mustUnderstand="1"
      id="B6432...466"
      actor="http://schemas.../next"
      messageID="uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
      timestamp="2112-03-15T11:12:12Z"/>

    <Provider providerID="example.com"
      affiliationID="affiliation.example.com"
      S:mustUnderstand="1"
      id="A9kendan...542"
      actor="http://schemas.../next"/>

    <!-- other header blocks, eg wsse:security, may go here -->

  </S:Header>

  <S:Body>

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

      <detail namespace="urn:liberty:sb:2003-08">
        <Status code="sb:"
          ref="Foo"
          comment="Bar" />
        <Correlation id="A13454...245"/>
      </detail>
    </S:Fault>

  </S:Body>

</S:Envelope>
```

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

5. Messaging-specific Header Blocks

The header blocks defined in this section implement the ID-* message exchange model. The first, the `<Correlation>` header block, addresses message correlation. The second, the `<Provider>` header block, addresses `providerID` and `affiliationID` declaration.

The messaging processing rules associated with these two header blocks are given in Section 5.3: 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. [LibertyInteract] defines a header block, for example.

5.1. Message Correlation: The <Correlation> Header Block

This section defines the SOAP-bound ID-* message correlation facilities.

5.1.1. The correlationType Header Block Type

The `correlationType` header block type provides a means for correlating SOAP-bound ID-* messages. Message correlation is achieved by inserting a `<Correlation>` element in SOAP-bound ID-* message headers and using the `messageID` attribute to identify individual messages. Additionally, a message may refer to another message by setting its `refToMessageID` attribute to the value of the `messageID` of the message of interest.

The `correlationType` header block type defines the following attributes:

- `messageID` [Required] – The unique ID of the message.
  Note:
  The messageID is intended to act as a *nonce*. Thus it is subject to specific processing rules defined below in Messaging Processing Rules , and the value-specific guidance given below in Section 5.1.2: `messageID` Value Requirements .

- `refToMessageID` [Optional] – A copy of the `messageID` attribute value of the message being responded to, or being correlated with in some application-specific fashion, if any.

- `timestamp` [Required] – The time the sender sent the message. Note guidance on time values given in Time Values above.

- `id` [Optional] – identifies a `<Correlation>` header block instance. This attribute MUST be used when the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the set of signed message components.

- `S:mustUnderstand` [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].

- `S:actor` [Optional] – The SOAP actor attribute [SOAPv1.1].

The schema fragment in Figure 4 defines the `Correlation` header block type.

```
<xs:complexType name="correlationType">
  <xs:attribute name="messageID" type="IDType" use="required"/>
  <xs:attribute name="refToMessageID" type="IDType" use="optional"/>
  <xs:attribute name="timestamp" type="xs:dateTime" use="required"/>
  <xs:attribute name="id" type="xs:ID" use="optional"/>
  <xs:attribute ref="S:mustUnderstand" use="optional"/>
  <xs:attribute ref="S:actor" use="optional"/>
</xs:complexType>
```

Figure 4. The `Correlation` Header Block Type Schema

5.1.2. messageID Value Requirements

Values of the `messageID` attribute of objects of type `correlationType` MUST satisfy the following property:

> Any party that assigns a value to a `messageID` 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 probability of two randomly chosen identifiers being identical MUST be less than 2^-128 and SHOULD be less than 2^-160. The above requirement MAY be met by applying Base64 [RFC2045] encoding to a randomly chosen value [RFC1750] 128 or 160 bits in length.

It is OPTIONAL for a `messageID` value to be resolvable in principle to some resource. In the case that a `messageID` is resolvable in principle (for example, it is in the form of a URI reference [RFC2396]), it is OPTIONAL for the identifier to be dereferenceable.

5.1.3. <Correlation> Header Block Element

The `<Correlation>` header block implements the features of the `correlationType` header block type, described above.

The schema fragment illustrated in Figure 5 defines the `<Correlation>` element, and Example 4 illustrates an instantiated one.

```
<xs:element name="Correlation" type="correlationType"/>
```

Figure 5. The `<Correlation>` Header Block Element Schema

```
<Correlation S:mustUnderstand="1"
  id="A13454...245"
  actor="http://schemas.../next"
  messageID="uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
  timestamp="2112-03-15T11:12:12Z"  />
```

Example 4. An Instantiated <Correlation> Header Block

5.2. Provider ID / Affiliation ID Declaration: The <Provider> Header Block

This section defines the <Provider> header block. This header block provides a means for a sender to claim that it is represented 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].

5.2.1. The ProviderType Header Block Type

The ProviderType header block type defines the following attributes:

- providerID [Required] – The Provider ID of the sender.
- affiliationID [Optional] – The Affiliation ID of the sender, if any.
- id [Optional] – identifies a <Provider> header block instance. This attribute MUST be used when the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the set of signed message components.
- S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

The schema fragment in Figure 6 defines the ProviderType header block type.

```
<xs:complexType name="ProviderType">
  <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
  <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
  <xs:attribute name="id" type="xs:ID" use="optional"/>
  <xs:attribute ref="S:mustUnderstand" use="optional"/>
  <xs:attribute ref="S:actor" use="optional"/>
</xs:complexType>
```

Figure 6. The ProviderType Header Block Type Schema

5.2.2. <Provider> Header Block Element

The <Provider> header block element implements the features of the providerType header block type.

The schema fragment shown in Figure 7 defines the <Provider> element, and Example 5 illustrates an instantiated <Provider> element.

```
<xs:element name="Provider" type="providerType"/>
```

Figure 7. The <Provider> Header Block Element Schema

```
<Provider providerID="example.com"
  affiliationID="affiliation.example.com"
  S:mustUnderstand="1"
  id="A9kendan...542"
  actor="http://schemas.../next"/>
```

Example 5. An instantiated <Provider> Header Block

5.3. Messaging Processing Rules

Overall processing of SOAP-bound ID-* messages, which by definition contain the <Correlation> and the <Provider> header blocks, 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 (the first nine are defined in this specification, the remainder are defined in the cited specifications):

- <Correlation>
- <Provider>
- <ProcessingContext>
- <Consent>
- <UsageDirective>
- <ServiceInstanceUpdate>
- <Timeout>

- `<CredentialsContext>`
- `<wsse:Security>` [LibertySecMech]
- `<is:UserInteraction>` [LibertyInteract]

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).

5.3.1. 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 include exactly one `<Correlation>` header block (see Section 5.1) in the `<S:Header>` child element of the `<S:Envelope>` element.
   The values of the `<Correlation>` header block attributes MUST be set as follows:
   A. `messageID` MUST be present, and its value SHOULD be set according to the rules presented in Section 5.1.2: `messageID` Value Requirements .
   B. `refToMessageID` MUST be present if the sender is both participating in a request-response MEP and is responding to a prior-received message. The `refToMessageID` attribute value MUST be set to the value of the `messageID` attribute from the prior-received message. Additionally, the `refToMessageID` value SHOULD be set according to the rules presented in Section 5.1.2: `messageID` Value Requirements .
   Note:
   If the outgoing message will convey an ID-* fault message (Section 4.4), then the `refToMessageID` attribute of its `<Correlation>` header block SHOULD NOT be present.
   C. `timestamp` MUST be present. Its value SHOULD be set to the time at which the message is sent. The `timestamp` value MUST conform to the rules presented in Section 2.5: Time Values
   D. `id` MUST be present if the sender is signing the message [LibertySecMech] and including the `<Correlation>` header block among the signed elements. Its value SHOULD be set according to the rules presented in Section 5.1.2: `messageID` Value Requirements
   E. `S:actor` SHOULD be present. Its value SHOULD be `"http://schemas.xmlsoap.org/soap/actor/next"`.
   F. `S:mustUnderstand` SHOULD be present. Its value SHOULD be `TRUE`.
3. The message SHOULD include exactly one, and MUST include no more than one `<Provider>` header block (see Section 5.2) in the `<S:Header>` child element of the `<S:Envelope>` element. If this header block is present, the sender is claiming to be a Liberty provider with the specified `providerID`.
   The values of the `<Provider>` header block attributes 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].

C. `id` MUST be present if the sender is signing the message [LibertySecMech] and including the `<Provider>` header block among the signed elements. Its value SHOULD be set according to the rules presented in Section 5.1.2: `messageID` Value Requirements

D. `S:actor` SHOULD be present. Its value SHOULD be `"http://schemas.xmlsoap.org/soap/actor/next"`.

E. `S:mustUnderstand` SHOULD be present. Its value SHOULD be `TRUE`.

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

5. 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.

6. 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 5.4: Correlation and Provider ID / Affiliation ID Examples .

5.3.2. 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 inbound SOAP-bound ID-* message.

1. Processing common to **all** received ID-* header blocks:

A. The `S:actor` attribute MAY be present. If present, its value SHOULD be `"http://schemas.xmlsoap.org/soap/actor/next"` or some other previously agreed upon (out-of-band) value.

B. The `S:mustUnderstand` attribute MAY be present. If present, its value SHOULD be `TRUE`.

C. A single `<Correlation>` header block should be present in the header of the message.

D. If the foregoing tests (1.A and 1.B and 1.C) hold true, processing continues with step 2.

E. Otherwise, the receiver MAY respond to the sender 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:
  - `"sb:InvalidActor"` if the failed test is 1.A,
  - `"sb:InvalidMustUnderstand"` if the failed test is 1.B,
  - `"sb:IDStarMsgNotUnderstood"` if the failed test is 1.C.
- and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

The `<S:Fault>` SHOULD contain a `<S:faultcode>` of `S:Client`.
Additionally, a header block element of the same type as the offending header block SHOULD be included in the ID-* Fault message's `"<detail>"` element, and the `id` attribute value of this header block element SHOULD be set to the same value as the `id` attribute of the offending header block in the incoming message.
The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

2. Processing specific to the `<Correlation>` header block:

A. Ensure that the value of the `timestamp` attribute is within an appropriate offset from local time expressed in UTC. Absent other guidance, a value of 5 minutes MAY be used.

B. For messages within the previous step's time window, ensure that the `messageID` attribute has not been seen before. If it has, then this message is likely a duplicate. Receivers SHOULD maintain a list of received `messageID` values that, at least, fall within the previous step's time window.

C. If the `refToMessageID` attribute is present, and its value is non-null, and if the receiver is participating in a request-response MEP with the sending party, then the value of the `refToMessageID` attribute SHOULD match the value of the `messageID` attribute of a message previously sent by the receiver to the sender of the now incoming message.

D. If the foregoing tests (2.A through 2.C) hold true, processing continues with step 3 .

E. 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:
  - `"sb:StaleMsg"` if the failed test is 2.A,
  - `"sb:DuplicateMsg"` if the failed test is 2.B,
  - `"sb:InvalidRefToMsgID"` if the failed test is 2.C,
- and a `ref` attribute with its value taken from the `messageID` value of the incoming message.

The <S:Fault> SHOULD contain a <S:faultcode> of S:Client.
Additionally, a header block element of the same type as the offending header block SHOULD be included in the ID-* Fault message's "s:detail" element, and the id attribute value of this header block element SHOULD be set to the same value as the id attribute of the offending header block in the incoming message.
The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

3. At this point, the receiver of the message MAY cease processing the message, and indicate to the requester that the message should be re-submitted to a different endpoint, according to the rules specified in Section 6.4.4.2

4. Processing specific to the <Provider> header block:

A. Verify that any declared providerID or affiliationID, are valid. The receiver SHOULD perform this verification and validation against metadata obtained via methods specified by [LibertyMetadata].

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

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

The <S:Fault> SHOULD contain a <S:faultcode> of S:Client.
Additionally, a header block element of the same type as the offending header block SHOULD be included in the ID-* Fault message's "<detail>" element, and the id attribute value of this header block element SHOULD be set to the same value as the id attribute of the offending header block in the incoming message.
The receiver MAY discard the incoming message. The receiver is finished processing this incoming message at this point.

5. Processing specific to ID-* header blocks **other than** <Correlation> and <Provider> – see appropriate sections in this and other specifications:

- For <wsse:Security>, see [LibertySecMech].
  Note:
  It should be noted that the receiver MAY return an sb-ext:InappropriateCredentials based on their processing of the <wsse:Security> header block, under conditions specified below in Section 6.4.4 and Section 6.3.4, in addition to other conditions such as an expired credential (see [LibertySecMech]).
- For <ProcessingContext>, see Section 6.1.
- For <Consent>, see Section 6.2.

- For `<UsageDirective>`, see Section 6.6.
- For `<ServiceInstanceUpdate>`, see Section 6.4.
- For `<Timeout>`, see Section 6.5.
- For `<CredentialsContext>`, see Section 6.3.
- For `<is:UserInteraction>`, see [LibertyInteract].

The manner of reporting any issues with these header blocks is specified in the respective section of this specification or in other specification(s) as noted above. If there are no issues with these header blocks, then processing continues with step 6 below, otherwise the receiver is finished processing this incoming message at this point.

6. 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, based on the ordinary ID-* message or ID-* fault message contained in the encompassing SOAP message's `<S:Body>` element.
   Or, if the message contained in the encompassing SOAP message's `<S:Body>` element is not recognizable as either an ordinary ID-* message or an ID-* fault message, and thus 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:
     - `"sb: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 6, and they are not further discussed in this document.

5.4. Correlation and Provider ID / Affiliation ID Examples

Example 6 illustrates a SOAP-bound ID-* message, containing both `<Correlation>` and `<Provider>` header blocks, and conveying a Personal Profile (ID-PP) Modify message [LibertyIDPP].

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

  <S:Header>

    <!-- other header blocks, eg wsse:security, may go here -->

    <Correlation S:mustUnderstand="1"
      id="A13454...245"
      actor="http://schemas.../next"
      messageID="uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
      timestamp="2112-03-15T11:12:12Z"/>

    <Provider providerID="example.com"
      affiliationID="affiliation.example.com"
      S:mustUnderstand="1"
      id="A9kendan...542"
      actor="http://schemas.../next"/>

    <!-- other header blocks, eg wsse:security, may go here -->

  </S:Header>

  <S:Body>

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

  </S:Body>

</S:Envelope>
```

Example 6. A SOAP-bound ID-* Message with a `<Correlation>` Header Block

Example 7 illustrates a SOAP-bound ID-* message, containing both `<Correlation>` and `<Provider>` header blocks. The `<Correlation>` header block is referring to the message in the previous example, which conveyed an ID-PP Modify message, and the message in Example 7 is conveying an ID-PP ModifyResponse message in response to the former. Note how the `refToMessageID` attribute is referencing the `messageID` in the example above. Additionally, note that the `<Provider>` header block is claiming that its Provider ID is `example579.com`, which is different than that claimed by the sender of the message in Example 6. Note that both system entities are claiming to belong to have the same Affiliation ID, `affiliation.example.com`.

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

  <S:Header>

    <!-- other header blocks, eg wsse:security, go here -->

    <Correlation S:mustUnderstand="1"
      id="B893483..83736"
      actor="http://schemas..../next"
      messageID="uuid:aaaaeeee-efef-fefe-efef-abababababab"
      refToMessageID=
      "uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
      timestamp="2112-03-15T11:12:13Z"/>

    <Provider providerID="example579.com"
      affiliationID="affiliation.example.com"
      S:mustUnderstand="1"
      id="A0brijas...975"
      actor="http://schemas.../next"/>

    <!-- other header blocks, eg wsse:security, go here -->

  </S:Header>

  <S:Body>

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

  </S:Body>

</S:Envelope>
```

Example 7. A SOAP-bound ID-* Message with a `<Correlation>` Header Block Referencing a Another Message

6. Optional Header Blocks

This section describes ID-* header blocks that are "optional" in the sense that they are not integral to basic ID-* messaging, per se. For example, the `<Correlation>` header block is minimally necessary to realize the simple one-way and request-reply MEPs defined above. Plus, the `<Provider>` header block facilitates receiver retrieval of provider metadata, which is also minimally necessary for communication between Liberty-enabled system entities.

In contrast, the header blocks described in this section are not strictly necessary to realize such basic messaging notions. However, the optional header blocks described below are essential to realizing several of the higher-level Project Liberty notions–for example: Principal consent and policy-based interactions.

The optional header blocks described in this specification are:

- `<ProcessingContext>`
- `<Consent>`
- `<UsageDirective>`
- `<ServiceInstanceUpdate>`
- `<Timeout>`
- `<CredentialsContext>`

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 5.3: 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 6.1.3: `<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 6.1.3.2.

6.1.1. The ProcessingContextType Header Block Type

The `ProcessingContextType` content model is `anyURI`. It defines the following attributes:

- `id` [Optional] – identifies a `<ProcessingContext>` header block instance. This attribute MUST be used when the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the set of signed message components.
- `S:mustUnderstand` [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].

- `S:actor` [Optional] – The SOAP actor attribute [SOAPv1.1].

The following schema fragment defines the `ProcessingContext` header block type:

```
<xs:complexType name="ProcessingContextType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Figure 8. The `ProcessingContext` Header Block Type Schema

6.1.2. <ProcessingContext> Header Block Element

The `<ProcessingContext>` schema element is given in Figure 9.

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

Figure 9. The <ProcessingContext> Element Schema

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

  <S:Header>

    <!-- other header blocks, eg wsse:security, may go here -->

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

    <Correlation S:mustUnderstand="1"
      id="B893483..83736"
      actor="http://schemas..../next"
      messageID="uuid:eeeecccc-efef-fefe-efef-bddbbddb"
      timestamp="2112-03-15T11:12:13Z"/>

    <Provider providerID="example.com"
      affiliationID="affiliation.example.com"
      S:mustUnderstand="1"
      id="A9kendan...542"
      actor="http://schemas.../next"/>

    <!-- other header blocks, eg wsse:security, may go here -->

  </S:Header>

  <S:Body>

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

  </S:Body>

</S:Envelope>
```

Example 8. A SOAP-bound ID-* Message with an Attached <ProcessingContext> Header Block

6.1.3. <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.

6.1.3.1. <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.

6.1.3.2. 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 (see [LibertyInteract]), 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.

6.1.3.3. 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.

6.1.3.4. Sender Processing Rules

A sender MAY include a `<ProcessingContext>` header block in a SOAP-bound ID-* message along with other header blocks besides `<Correlation>` and `<Provider>` such as `<UserInteraction>`. 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.

6.1.3.5. 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 honour 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 (see [LibertyInteract] 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:
  - `"sb: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:
  - `"sb: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 consentType Header Block Type

The `<Consent>` header block element MAY be employed by either a requester 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 `consentType` header block type 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.
- `id` [Optional] – identifies a `<Consent>` header block instance. This attribute MUST be used when the message is signed as described in [LibSecMech], and the element instance is to be included in the signed message components.
- `S:mustUnderstand` [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- `S:actor` [Optional] – The SOAP actor attribute [SOAPv1.1].

The schema fragment in Figure 10 defines the `Consent` header block type.

```
<xs:complexType name="consentType">
<xs:attribute name="uri" type="xs:anyURI" use="required"/>
<xs:attribute name="timestamp" type="xs:dateTime" use="optional"/>
<xs:attribute name="id" type="xs:ID" use="optional"/>
<xs:attribute ref="S:mustUnderstand" use="optional"/>
<xs:attribute ref="S:actor" use="optional"/>
</xs:complexType>
```

Figure 10. The `Consent` Header Block Type Schema

6.2.2. <Consent> Header Block Element

The schema fragment in Figure 11 defines the `<Consent>` element:

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

Figure 11. The `<Consent>` Element Schema

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

   <S:Header>

      <!-- other header blocks, eg wsse:security, may go here -->

      <Consent id="A124395732495743"
         uri="urn:liberty:consent:obtained"
         timestamp="2112-03-15T11:12:10Z"/>

      <Correlation S:mustUnderstand="1"
         id="B893483..83736"
         actor="http://schemas..../next"
         messageID="uuid:eeeecccc-efef-fefe-efef-bddbbddb"
         timestamp="2112-03-15T11:12:13Z"/>

      <!-- other header blocks, eg wsse:security, may go here -->

   </S:Header>

   <S:Body>

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

   </S:Body>

</S:Envelope>
```

Example 9. A SOAP-bound ID-* Message with an Attached <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 requester 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:

- lib:RequestAuthnContext [Optional] – a container that allows the expression of a requested authentication context (see [LibertyAuthnContext]).
- SecurityMechID [Optional] – A set of elements that specify ID-WSF security mechanism URIs (see [Liberty-SecMech]).
- id [Optional] – An attribute facilitating references to elements of this type.
- S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

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

```
<xs:complexType name="CredentialsContextType">
   <xs:sequence>
      <xs:element ref="lib:RequestAuthnContext" minOccurs="0"/>
      <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="optional"/>
   <xs:attribute ref="S:mustUnderstand" use="optional"/>
   <xs:attribute ref="S:actor" use="optional"/>
</xs:complexType>

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

Figure 12. The <CredentialsContext> Header Block Element and Type Schema

6.3.3. CredentialsContext Example

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

  <S:Header>

    <sb:Correlation S:mustUnderstand="1"
      messageID="uuid:0023923-28329023-238239023"
      refToMessageID="uuid:00123-ad32-23-09923-88239"
      timestamp="2003-06-06T12:10:10Z"  />

      <!-- Says that the sender would like credentials that include RequestAuthnContext
          as specified -->

      <sb-ext:CredentialsContext mustUnderstand="1">

         <lib:RequestAuthnContext>
         ...
         </lib:RequestAuthnContext>

      </sb-ext: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 namespace="urn:liberty:sb:2003-08">
        <Status code="sb-ext:InappropriateCredentials"/>
        <wsse:Security id="a6352...564"/>
      </detail>
    </S:Fault>
  </S:Body>

</S:Envelope>
```

Example 10. A CredentialsContext Header Offered in Response to a Request with Inappropriate Credentials.

6.3.4. 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. The receiver SHOULD use the highest-listed `SecurityMechID` that it supports in future requests to the sender of this header.

6.4. The <ServiceInstanceUpdate> 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 <ServiceInstanceUpdate> allows a message receiver to indicate that a new SOAP endpoint, new credentials, or new security mechanisms should be employed by the sender of the message. This header block may be used in conjunction with the <sb-ext:InappropriateCredentials> and <sb-ext:EndpointMoved> faults, to indicate that the current message processing failed for those reasons, and should be submitted with the changes noted in any accompanying <ServiceInstanceUpdate> 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. ServiceInstanceUpdate Type and Element

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

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

- SecurityMechID [Optional] – a URI indicating a new security mechanism that should be used in place of those currently employed in the sending of this message.
- Credential [Optional] – a container for arbitrary credentials that are supplied by the message sender. This contains an optional notOnOrAfter attribute, used to indicate an expiration time for the supplied credential.
- id [Optional] – An attribute facilitating references to elements of this type.
- S:mustUnderstand [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- S:actor [Optional] – The SOAP actor attribute [SOAPv1.1].

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

```
<xs:complexType name="ServiceInstanceUpdateType">
   <xs:sequence>
      <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Credential" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
           <xs:sequence>
              <xs:any namespace="##any" processContents="lax"/>
           </xs:sequence>
           <xs:attribute name="notOnOrAfter" type="xs:dateTime" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Endpoint" type="xs:anyURI" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="optional"/>
   <xs:attribute ref="S:mustUnderstand" use="optional"/>
   <xs:attribute ref="S:actor" use="optional"/>
</xs:complexType>

<xs:element name="ServiceInstanceUpdate" type="ServiceInstanceUpdateType"/>
```

Figure 13. The <ServiceInstanceUpdate> Header Block Element and Type Schema

6.4.3. ServiceInstanceUpdate 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:2003-08"
  xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>
    <sb:Correlation S:mustUnderstand="1"
      messageID="uuid:0023923-28329023-238239023"
      refToMessageID="uuid:00123-ad32-23-09923-88239"
      timestamp="2003-06-06T12:10:10Z"  />

    <sb-ext:ServiceInstanceUpdate mustUnderstand="1">

      <sb-ext:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</sb-ext:SecurityMe
chID>

      <sb-ext:Credential notOnOrAfter="2003-06-06T15:10:10Z">

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

      </sb-ext:Credential>

    </sb-ext:ServiceInstanceUpdate>

  </S:Header>

  <S:Body>

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

  </S:Body>

</S:Envelope>
```

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

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

  <S:Header>
    <sb:Correlation S:mustUnderstand="1"
      messageID="uuid:3736251-21532023-774625253"
      refToMessageID="uuid:0023923-28329023-238239023"
      timestamp="2003-06-06T12:15:04Z" />

    <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 11. A ServiceInstanceUpdate 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:2003-08">

  <S:Header>
    <sb:Correlation S:mustUnderstand="1"
      messageID="uuid:0023923-28329023-238239023"
      refToMessageID="uuid:00123-ad32-23-09923-88239"
      timestamp="2003-06-06T12:10:10Z"  />

    <sb-ext:ServiceInstanceUpdate mustUnderstand="1">
      <sb-ext:Endpoint>http://example.com:443/soap</sb-ext:Endpoint>
    </sb-ext:ServiceInstanceUpdate>

  </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 namespace="urn:liberty:sb:2003-08">
        <Status code="sb-ext:EndpointMoved"/>
      </detail>

    </S:Fault>

  </S:Body>

</S:Envelope>
```

Example 12. A ServiceInstanceUpdate Specifying an Updated Endpoint.

6.4.4. Processing Rules

6.4.4.1. Processing Rules for the ServiceInstanceUpdate header

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

The sender of this header block MUST include at least one of the optional elements contained in the header block (`<Endpoint>`, `<SecurityMechID>`, or `<Credential>`).

Any `<Endpoint>` specified SHOULD be set to a complete URI.

This header block may contain one or more `<SecurityMechID>` elements. In this case, the service is stating that the security mechanisms denoted by these elements SHOULD be used in preference to any that had previously been specified (for example, upon registration of the service at a discovery service [LibertyDisco]). Any mechanisms

supplied SHOULD be listed in order of preference by the service. `<SecurityMechID>` elements MUST be specified according to [LibertySecMech].

If the message sender includes one or more `<SecurityMechID>` elements in their response, that were not previously specified (for example during registration at a discovery service), then they MUST supply appropriate credentials for the message recipient to use in accordance with the supplied mechanisms. This implies, for example, that if a bearer (see [LibertySecMech]) mechanism is supplied in this header block, but was not previously specified for the requester, that the sender should supply a bearer token that the requester may use with this mechanism.

The `<Credential>` element of the header block may contain a `notOnOrAfter` attribute, which the sender SHOULD use to indicate the latest time at which the supplied credential is valid, unless the credential being supplied contains some other indication of such a value.

The entity constructing a `<Credential>` element SHOULD specify a valid credential for use by the recipient. For example, if the token constructed is of type `wsse:BinarySecurityToken` then it should conform to the rules specified in [LibertySecMech] and supporting specifications.

6.4.4.2. Processing Rules for the EndpointMoved SOAP Fault

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 a `<ServiceInstanceUpdate>` header, indicating the new endpoint which should be used to re-submit this message, and any further messages directed to the responding service.

If the receiver of this fault response also received a `<ServiceInstanceUpdate>` header in the response, it MAY re-submit the failed request to any endpoint specified in that header, but it SHOULD provide a different `messageID` in the `<Correlation>` header block of 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 requester wishes the receiver to spend in processing their request
- `id` [Optional] – An attribute facilitating references to elements of this type.
- `S:mustUnderstand` [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- `S:actor` [Optional] – The SOAP actor attribute [SOAPv1.1].

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

```
<xs:complexType name="TimeoutType">
   <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
   <xs:attribute name="id" type="xs:ID" use="optional"/>
   <xs:attribute ref="S:mustUnderstand" use="optional"/>
   <xs:attribute ref="S:actor" use="optional"/>
</xs:complexType>

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

Figure 14. The <Timeout> Header Block Element and Type Schema

6.5.3. Timeout Example

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

  <S:Header>

    <sb-ext:Timeout mustUnderstand="1" id="timeout.123"
      maxProcessingTime="7"/>

  </S:Header>

  <S:Body>

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

  </S:Body>

</S:Envelope>
```

Example 13. Example of a Request with Timeout Specified

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

  <S:Header>

    ...

  </S:Header>

  <S:Body>

    <S:Fault>

      <S:faultcode>
        S:Server
      </S:faultcode>

      <S:detail>

        <sb:Status code="sb-ext:ProcessingTimeout"/>

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

        <sb-ext:Timeout id="timeout.123"/>

    </S:Fault>

  </S:Body>

</S:Envelope>
```

Example 14. Example of a Timed-out Response

6.5.4. 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 SHOULD respond with a SOAP Fault with a `code` of `sb-ext: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 `sb-ext: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 Header 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.
- `id` [Optional} – An attribute facilitating references to elements of this type.
- `S:mustUnderstand` [Optional] – The SOAP mustUnderstand attribute [SOAPv1.1].
- `S:actor` [Optional] – The SOAP actor attribute [SOAPv1.1].
- `<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 fragments in Figure 15 and Figure 16 defines the `<UsageDirective>` header type and element.

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

Figure 15. The `<UsageDirective>` Header Block Element Schema

```
<complexType name="UsageDirectiveType">
  <sequence>
    <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="ref" type="reference" use="required"/>
  <attribute name="id" type="id" use="optional"/>
  <attribute ref="S:mustUnderstand" use="optional"/>
  <attribute ref="S:actor" use="optional"/>
</complexType>
```

Figure 16. The `UsageDirective` Header Block Type Schema

6.6.3. Usage Directive Examples

Example 15 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://circle-of-trust.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:2003-08"
  xmlns:pp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    <!-- other header blocks, eg wsse:security, may go here -->

    <UsageDirective
      id="directive1000"
      ref="#datarequest001"
      S:mustUnderstand="1">

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

    </UsageDirective>

    <Correlation S:mustUnderstand="1"
      id="A13454...245"
      S:actor="http://schemas.../next"
      messageID="uuid:efefefef-eeee-ffff-cccc-ffffffffbbbb"
      timestamp="2112-03-15T11:12:14Z"/>

    <!-- other header blocks, eg wsse:security, may go here -->

  </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 15. A Usage Directive on a Request for the Address of a Principal.

6.6.4. 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].

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="TRUE" MUST respond with a <S:Fault>. The s:Fault MUST contain a <S:Detail> element which in turn MUST contain a <Status> element with its <StatusCode> set to sb: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 honour a non-mandatory (with mustUnderstand="FALSE") <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 <StatusCode> set to sb: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 honoured.

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 honour. The ref attribute of these headers SHOULD be set to the messageID of the <Correlation> element.

7. 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.

8. 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.

Bibliography

Normative

[LibertyDST] "Liberty ID-WSF Data Services Template Specification," Version 1.1, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs* Kainulainen, Jukka, Ranganathan, Aravindan, eds.

[LibertyIDFFOverview] Wason, Thomas, eds. "Liberty ID-FF Architecture Overview," Version 1.2-errata-v1.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 1.1, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.1, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertyAuthnContext] Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 1.3, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.2, Liberty Alliance Project (14 December 2004). *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 [November 1996]*.

[RFC2119] Bradner, S., eds. "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 [March 1997]*.

[RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2396.txt [August 1998]*.

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

[SAMLGloss] Hodges, Jeff, Maler, Eve, eds. (05 November 2002). "Glossary for the OASIS Security Assertion Markup Language (SAML)," Version 1.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/security/#documents*

[Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-1/*

[Schema2] Biron, Paul V., Malhotra, Ashok, eds. (May 2002). "XML Schema Part 2: Datatypes," 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, eds. (Oct 2000). "Extensible Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/2000/REC-xml-20001006*

[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*

Informational

[LibertyDisco] Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification," Version 1.2, Liberty Alliance Project (12 December 2004). *http://www.projectliberty.org/specs*

[LibertyGlossary] "Liberty Technical Glossary," Version 1.4, Liberty Alliance Project (14 Dec 2004). *http://www.projectliberty.org/specs* Hodges, Jeff, eds.

[LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Overview," Version 1.1, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertyIDPP] Kellomaki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

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

[RFC1750] Eastlake , D., Crocker, S., Schiller, J., eds. (December 1994). "Randomness Recommendations for Security ," RFC 1750, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc1750.txt [August 2003]*.

[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 [June 1999]*.

[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 Proposed Recommendation (07 May 2003). *http://www.w3.org/TR/2003/PR-soap12-part1-20030507/*

A. Liberty ID-WSF SOAP Binding Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:sb:2003-08"
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:sb="urn:liberty:sb:2003-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:liberty:sb:2003-08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="1.0-02">

   <!-- Filename: lib-arch-soap-binding.xsd  -->
   <!-- $Id: lib-arch-soap-binding.xsd,v 1.8.4.1 2005/01/25 18:39:11 dchampagne Exp $ -->
   <!-- Author: Jeff Hodges -->
   <!-- Last editor: $Author: dchampagne $ -->
   <!-- $Date: 2005/01/25 18:39:11 $ -->
   <!-- $Revision: 1.8.4.1 $ -->

   <xs:import
      namespace="http://schemas.xmlsoap.org/soap/envelope/"
      schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>

   <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>

   <xs:annotation>
      <xs:documentation>
        Liberty ID-WSF SOAP Binding Specification XSD
      </xs:documentation>
      <xs:documentation>

The source code in this XSD file was excerpted verbatim from:

Liberty ID-WSF SOAP Binding Specification
Version 1.0
12th November 2003

Copyright (c) 2003-2005 Liberty Alliance participants, see
http://www.projectliberty.org/specs/idwsf_copyrights.html

      </xs:documentation>
   </xs:annotation>

   <!-- message correlation header block -->

   <xs:complexType name="CorrelationType">
      <xs:attribute name="messageID" type="IDType" use="required"/>
      <xs:attribute name="refToMessageID" type="IDType" use="optional"/>
      <xs:attribute name="timestamp" type="xs:dateTime" use="required"/>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </xs:complexType>

   <xs:element name="Correlation" type="CorrelationType"/>

   <!-- provider= header block  -->

   <xs:complexType name="ProviderType">
      <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
      <xs:attribute name="affiliationID" type="xs:anyURI" use="optional"/>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </xs:complexType>

   <xs:element name="Provider" type="ProviderType"/>
```

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

   <xs:complexType name="ProcessingContextType">
      <xs:simpleContent>
         <xs:extension base="xs:anyURI">
            <xs:attribute name="id" type="xs:ID" use="optional"/>
            <xs:attribute ref="S:mustUnderstand" use="optional"/>
            <xs:attribute ref="S:actor" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

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

   <!-- consent claim 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:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </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="id" type="xs:ID" use="optional"/>
      <xs:attribute name="ref" type="xs:IDREF" use="required"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </xs:complexType>

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

</xs:schema>
```

B. Liberty ID-WSF SOAP Binding Extension Schema (April 2004)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:sb:2004-04"
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:sb-ext="urn:liberty:sb:2004-04"
   xmlns:lib="urn:liberty:iff:2003-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:liberty:sb:2004-04"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

   <!-- Author: John Kemp -->
   <!-- Last editor: $Author: dchampagne $ -->
   <!-- $Date: 2005/01/25 18:39:11 $ -->
   <!-- $Revision: 1.3.2.1 $ -->

   <xs:import
      namespace="http://schemas.xmlsoap.org/soap/envelope/"
      schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>

   <xs:import
      namespace="urn:liberty:iff:2003-08"
      schemaLocation="liberty-idff-protocols-schema-1.2-errata-v3.0.xsd"/>

   <xs:include schemaLocation="liberty-idwsf-utility-v1.1.xsd"/>

   <xs:annotation>
      <xs:documentation>
         Liberty ID-WSF SOAP Binding Specification Extension XSD
      </xs:documentation>
      <xs:documentation>
         The source code in this XSD file was excerpted verbatim from:

         Liberty ID-WSF SOAP Binding Specification
         Version 1.2
         December 2004

         Copyright (c) 2004-2005 Liberty Alliance participants, see
         http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php

      </xs:documentation>
   </xs:annotation>

   <xs:complexType name="CredentialsContextType">
      <xs:sequence>
         <xs:element ref="lib:RequestAuthnContext" minOccurs="0"/>
         <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </xs:complexType>

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

   <xs:complexType name="ServiceInstanceUpdateType">
      <xs:sequence>
         <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="Credential" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                 <xs:any namespace="##any" processContents="lax"/>
              </xs:sequence>
              <xs:attribute name="notOnOrAfter" type="xs:dateTime" use="optional"/>
            </xs:complexType>
         </xs:element>
```

```
         <xs:element name="Endpoint" type="xs:anyURI" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </xs:complexType>

   <xs:element name="ServiceInstanceUpdate" type="ServiceInstanceUpdateType"/>

   <xs:complexType name="TimeoutType">
      <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute ref="S:mustUnderstand" use="optional"/>
      <xs:attribute ref="S:actor" use="optional"/>
   </xs:complexType>

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

</xs:schema>
```

C. Liberty ID-WSF Utility Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- filename: liberty-idwsf-utility-v1.1.xsd -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
   <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.

         Copyright 2003-2005 Liberty Alliance Project, see
         http://www.projectliberty.org/specs/idwsf_1_1_copyrights.php

         This file intended for inclusion, rather than importation,
         into other schemas.

         This version: 2004-12-14

      </xs:documentation>
   </xs:annotation>
   <xs:simpleType name="IDType">
      <xs:annotation>
         <xs:documentation>This type should be used to provided 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:element name="Status" type="StatusType">
      <xs:annotation>
         <xs:documentation> A standard Status type</xs:documentation>
      </xs:annotation>
   </xs:element>
   <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:QName" use="required"/>
      <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
      <xs:attribute name="comment" type="xs:string" use="optional"/>
   </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. SOAP Envelope Schema Listing

```
<?xml version='1.0' encoding='UTF-8' ?>

<!-- Schema for the SOAP/1.1 envelope

     This schema has been produced using W3C's SOAP Version 1.2 schema
     found at:

     http://www.w3.org/2001/06/soap-envelope

     Copyright 2001 Martin Gudgin, Developmentor.

     Changes made are the following:
     - reverted namespace to http://schemas.xmlsoap.org/soap/envelope/
     - reverted mustUnderstand to only allow 0 and 1 as lexical values
   - made encodingStyle a global attribute 20020825

   Further changes:

   - removed default value from mustUnderstand attribute declaration - 20030314

     Original copyright:

     Copyright 2001 W3C (Massachusetts Institute of Technology,
     Institut National de Recherche en Informatique et en Automatique,
     Keio University). All Rights Reserved.
     http://www.w3.org/Consortium/Legal/

     This document is governed by the W3C Software License [1] as
     described in the FAQ [2].

     [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
     [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD
-->


<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
   targetNamespace="http://schemas.xmlsoap.org/soap/envelope/" >


   <!-- Envelope, header and body -->

   <xs:element name="Envelope" type="tns:Envelope" />

   <xs:complexType name="Envelope" >
      <xs:sequence>
         <xs:element ref="tns:Header" minOccurs="0" />
         <xs:element ref="tns:Body" minOccurs="1" />
         <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax" />
   </xs:complexType>


   <xs:element name="Header" type="tns:Header" />

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


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

```
2086
2087    <xs:complexType name="Body" >
2088       <xs:sequence>
2089          <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
2090       </xs:sequence>
2091       <xs:anyAttribute namespace="##any" processContents="lax" >
2092          <xs:annotation>
2093             <xs:documentation>
2094                Prose in the spec does not specify that attributes are allowed on the Body element
2095  </xs:documentation>
2096          </xs:annotation>
2097  </xs:anyAttribute>
2098    </xs:complexType>
2099
2100
2101    <!-- Global Attributes.  The following attributes are intended to be usable via qualified
2102 attribute names on any complex type referencing them.  -->
2103
2104    <xs:attribute name="mustUnderstand" >
2105       <xs:simpleType>
2106          <xs:restriction base='xs:boolean'>
2107             <xs:pattern value='0|1' />
2108          </xs:restriction>
2109       </xs:simpleType>
2110    </xs:attribute>
2111
2112    <xs:attribute name="actor" type="xs:anyURI" />
2113
2114    <xs:simpleType name="encodingStyle" >
2115       <xs:annotation>
2116          <xs:documentation>
2117             'encodingStyle' indicates any canonicalization conventions followed in the contents of
2118 the containing element.  For example, the value 'http://schemas.xmlsoap.org/soap/encoding/'
2119 indicates the pattern described in SOAP specification
2120          </xs:documentation>
2121  </xs:annotation>
2122       <xs:list itemType="xs:anyURI" />
2123    </xs:simpleType>
2124
2125    <xs:attribute name="encodingStyle" type="tns:encodingStyle" />
2126
2127
2128    <xs:attributeGroup name="encodingStyle" >
2129       <xs:attribute ref="tns:encodingStyle" />
2130    </xs:attributeGroup>
2131
2132
2133    <xs:element name="Fault" type="tns:Fault" />
2134
2135    <xs:complexType name="Fault" final="extension" >
2136       <xs:annotation>
2137          <xs:documentation>
2138             Fault reporting structure
2139          </xs:documentation>
2140  </xs:annotation>
2141       <xs:sequence>
2142          <xs:element name="faultcode" type="xs:QName" />
2143          <xs:element name="faultstring" type="xs:string" />
2144          <xs:element name="faultactor" type="xs:anyURI" minOccurs="0" />
2145          <xs:element name="detail" type="tns:detail" minOccurs="0" />
2146       </xs:sequence>
2147    </xs:complexType>
2148
2149    <xs:complexType name="detail">
2150       <xs:sequence>
2151          <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax" />
2152       </xs:sequence>
```

```
      <xs:anyAttribute namespace="##any" processContents="lax" />
    </xs:complexType>

</xs:schema>
```