Liberty ID-WSF Authentication Service and Single Sign-On Service Specification

Version: v2.0-09

Editors:
Jeff Hodges, NeuStar, Inc.
Robert Aarts, Trustgenix, Inc.
Paul Madsen, NTT

Contributors:
Conor Cahill, America Online, Inc.
Darryl Champagne, IEEE-ISTO
Gary Ellison, Sun Microsystems, Inc.
Greg Whitehead, Trustgenix, Inc.

Abstract:

This specification defines an ID-WSF Authentication Protocol based on a profile of the Simple Authentication and Security Layer (SASL) framework mapped onto ID-* SOAP-bound messages. Next, it defines an ID-WSF Authentication Service which Identity Providers may offer. This service is based on the authentication protocol. The authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate with Identity Providers, using various authentication mechanisms, and obtain ID-WSF security tokens. Finally, it defines an ID-WSF Single Sign-On Service. This service provides authentication assertions to Web Service Consumers via a profile of the ID-FF Single Sign-On Protocol, enabling Web Service Consumers to interact with ID-FF-based, or other, Service Providers.

Filename: draft-liberty-idwsf-authn-svc-v2.0-09.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.

Copyright © 2005 Adobe Systems; America Online, Inc.; American Express Company; Amsoft Systems Pvt Ltd.;
Avatier Corporation; Axalto; Bank of America Corporation; BIPAC; BMC Software, Inc.; Computer Associates
International, Inc.; DataPower Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.;
Ericsson; Fidelity Investments; Forum Systems, Inc.; France Télécom; French Government Agence pour le
développement de l’administration électronique (ADAE); Gamefederation; Gemplus; General Motors; Giesecke &
Devrient GmbH; GSA Office of Governmentwide Policy; Hewlett-Packard Company; IBM Corporation; Intel
Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard International; Mobile Telephone Networks (Pty)
Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon Telegraph and Telephone Corporation; Nokia
Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle Corporation; Ping Identity Corporation;
Reactivity Inc.; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp Laboratories of America;
Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Supremacy Financial Corporation; Symlabs, Inc.;
Telecom Italia S.p.A.; Telefónica Móviles, S.A.; Trusted Network Technologies; Trustgenix; UTI; VeriSign, Inc.;
Vodafone Group Plc.; Wave Systems Corp. All rights reserved.
## Contents

1. Introduction ........................................................................................................... 4
2. Notation and Conventions ..................................................................................... 5
   2.1. Requirements Keywords .................................................................................. 5
   2.2. XML Namespaces .............................................................................................. 5
3. Terminology ............................................................................................................. 6
4. Authentication Protocol .......................................................................................... 9
   4.1. Conceptual Model .............................................................................................. 9
   4.2. Schema Declarations ....................................................................................... 9
   4.3. SOAP Header Blocks and SOAP Binding ....................................................... 9
   4.3.1. SOAP Binding ............................................................................................ 9
   4.4. SASL Profile Particulars .................................................................................. 9
   4.4.1. SASL "Service Name" ............................................................................... 10
   4.4.2. Composition of SASL Mechanism Names ................................................. 10
5. Authentication Service .......................................................................................... 10
   5.1. Conceptual Model ........................................................................................... 20
       5.1.1. Stipulating a Particular Authentication Context ..................................... 20
   5.2. Service Type Declaration ............................................................................... 20
   5.3. Rules for Authentication Service Providers ................................................... 21
   5.4. Rules for Authentication Service Consumers ............................................... 22
   5.5. Authentication Service Interaction Example ................................................ 23
6. Single Sign-On Service ......................................................................................... 26
   6.1. Conceptual Model ........................................................................................... 26
   6.2. Service Type Declaration ............................................................................... 27
   6.3. Rules for SSO Service Providers ..................................................................... 27
   6.4. Rules for SSO Service Consumers .................................................................. 28
7. Identity Token Mapping Service ........................................................................... 29
8. Password Transformations: The `PasswordTransforms` Element ...................... 31
9. Acknowledgments ................................................................................................. 33
10. References ........................................................................................................... 34
A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms ....... 37
B. Password Transformations .................................................................................... 39
   1. Truncation ............................................................................................................ 39
   2. Lowercase .......................................................................................................... 39
   3. Uppercase ........................................................................................................... 39
   4. Select .................................................................................................................. 39
C. `liberty-idwsf-authn-svc-v2.0.xsd` Schema Listing ............................................. 41
D. `liberty-idwsf-utility-v2.0.xsd` Schema Listing .................................................. 44
E. `liberty-utility-v2.0.xsd` Schema Listing .............................................................. 45
1. Introduction

The Simple Object Access Protocol (SOAP) specifications, [SOAPv1.1] and [SOAPv1.2], define an XML-based messaging paradigm, but do not specify any particular security mechanisms. They do not, in particular, describe how one SOAP node may authenticate with another SOAP node via an exchange of SOAP messages. Thus it is left to SOAP-based web services frameworks to provide their own notions of security, such as defining how authentication is accomplished.

This specification defines how to perform general identity authentication [WooLam92], also known as peer entity authentication [RFC2828], over SOAP, in the context of the Liberty Identity Web Services Framework (ID-WSF) [LibertyIDWSFOverview]. Rather than specify the particulars of one or more authentication mechanisms directly in this specification, we profile the Simple Authentication and Security Layer (SASL) framework [RFC2222].

SASL is an approach to modularizing protocol design such that the security design components, e.g. authentication and security layer mechanisms, are reduced to a uniform abstract interface. This facilitates a protocol’s use of an open-ended set of security mechanisms, as well as a so-called "late binding" between implementations of the protocol and the security mechanisms’ implementations. This late binding can occur at implementation- and/or deployment-time.

The SASL specification also defines how one packages authentication and security layer mechanisms to fit into the SASL framework, where they are known as SASL mechanisms, as well as register them with the Internet Assigned Numbers Authority (IANA) [IANA] for reuse.

This specification is organized as follows. First, it defines the ID-WSF Authentication Protocol. Next, it defines an ID-WSF Authentication Service Identity Providers may offer, which is based on the authentication protocol.

This authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate with Identity Providers using various authentication mechanisms and obtain ID-WSF security tokens.

Finally, it defines an ID-WSF Single Sign-On Service. This service provides authentication assertions to Web Services Consumers via a profile of the ID-FF Single Sign-On Protocol, enabling Web Services Consumers to interact with ID-FF-based Service Providers.
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.

2.1. Requirements Keywords

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.

2.2. XML Namespaces

This specification uses the XML namespace prefixes listed in Table 1.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac:</td>
<td>Represents the Liberty namespace defined in [LibertyAuthnContext]</td>
</tr>
<tr>
<td>disco:</td>
<td>Represents the namespace defined in [LibertyDisco].</td>
</tr>
<tr>
<td>lib:</td>
<td>Represents the namespace defined in [LibertyProtSchema].</td>
</tr>
<tr>
<td>pp:</td>
<td>Represents the namespace defined in [LibertyIDPP].</td>
</tr>
<tr>
<td>s:</td>
<td>Represents the SOAP namespace: <a href="http://www.w3.org/2001/12/soap-envelope">http://www.w3.org/2001/12/soap-envelope</a>, defined in [SOAPv1.1].</td>
</tr>
<tr>
<td>saml:</td>
<td>Represents the SAML V1.1 Assertion namespace defined in [SAMLCore11]</td>
</tr>
<tr>
<td>saml2:</td>
<td>Represents the SAML V2.0 Assertion namespace defined in [SAMLCore2]</td>
</tr>
<tr>
<td>samlp2:</td>
<td>Represents the SAML V2.0 Protocol namespace defined in [SAMLCore2]</td>
</tr>
<tr>
<td>sb:</td>
<td>Represents the Liberty namespace defined in [LibertySOAPBinding]</td>
</tr>
<tr>
<td>xs:</td>
<td>Represents the W3C XML schema namespace (<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>) defined in [Schema1].</td>
</tr>
</tbody>
</table>
This section defines key terminology used in this specification. Definitions for these, as well as other Liberty-specific terms, may also be found in [LibertyGlossary]. Note that the definition of some terms below differ slightly from the definition given in [LibertyGlossary]. For example see the definitions for client and server. This is because in such cases, the definition given in [LibertyGlossary] is a more general one, and the definition given here is a narrower one, specific to the context of this specification. See also [RFC2828] for overall definitions of security-related terms, in general. Other specific references are also cited below.

authentication

Authentication is the process of confirming a system entity's asserted identity with a specified, or understood, level of confidence [TrustInCyberspace].

authentication assertion

A SAML assertion typically consisting of a single <AuthenticationStatement>. The assertion issuer is stating that the subject of the assertion authenticated with it at some point in time. Assertions are typically time-limited [SAMLCore11].

authentication exchange

See authentication protocol exchange.

authentication mechanism

An authentication mechanism is a particular, identifiable, process or technique that results in a confirmation of a system entity's asserted identity with a specified, or understood, level of confidence.

authentication protocol exchange

Authentication protocol exchange is the term used in [RFC2222] to refer to the sequence of messages exchanged between the client and server as specified and governed by a particular SASL mechanism being employed to effect an act of authentication.

authentication server

The precise, specific role played by a server in the protocol message exchanges defined in this specification.

Authentication Service (AS)  Short form of "ID-WSF Authentication Service". The AS is a discoverable ID-WSF service.

Authentication Service Consumer  A Web Service Consumer (WSC) implementing the client-side of the ID-WSF Authentication Protocol (which is defined in this specification).

Authentication Service Provider (AS Provider)  A Web Service Provider (WSP) implementing the server-side of the ID-WSF Authentication Service defined in this specification (Section 5: Authentication Service).

client

A role assumed by a system entity who either explicitly or implicitly initiates an authentication exchange [RFC2828]. Client is implicitly defined in [RFC2222]. Also known as a SASL client.

disposable

A discoverable "in principle" service is one having a service type URI assigned (this is typically in done in the specification defining the service). A discoverable "in practice" service is one that is registered in some discovery service instance.

ID-WSF services are by definition discoverable "in principle" because such services are assigned a service type URI facilitating their registration in Discovery Service instances.

final SASL response

The final <SASLResponse> message sent from the server to the client in an authentication exchange.
An **ID-WSF Endpoint Reference** is a reference to a service instance. It contains the address, security context, and other metadata necessary for contacting the identified service instance. The underlying structure of an ID-WSF EPR is based on the `wsa:EndpointReference` of [WSAv1.0-SOAP] [WSAv1.0].

A [RFC2222] term referring to authentication exchange data sent by the client in the initial SASL request. It is used by a subset of SASL mechanisms. See Section 5.1 of [RFC2222].

The initial `<SASLRequest>` message sent from the client to the server in an authentication exchange.

A Web Service Consumer (WSC) that may or may not also be a Liberty-enabled User Agent or Device.

A process or technique for achieving a result [Merriam-Webster].

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.

A system entity which sends a service request to a provider.

A function or part performed, especially in a particular operation or process [Merriam-Webster].

A SASL mechanism is an authentication mechanism that has been profiled for use in the context of the SASL framework [RFC2222]. See [RFC2444] for a particular example of profiling an existing authentication mechanism—one-time passwords [RFC2289]—for use in the SASL context. SASL mechanisms are "named"; Mechanism names are listed in the column labeled as "MECHANISMS" in [SASLReg] (a copy of this registry document is reproduced in Appendix A for informational convenience; implementors should always fetch the most recent revision directly from [IANA]).

A role donned by a system entity which is intended to engage in defined exchanges with clients. This term is implicitly defined in [RFC2222] and in this specification is always synonymous with authentication server.

The physical instantiation of a service. A service instance is a web service at a distinct endpoint.

(1)A role donned by system entities. In the Liberty architecture, Service Providers interact with other system entities primarily via vanilla HTTP.

(2) From a Principal’s perspective, a Service Provider is typically a website providing services and/or goods.

A [SOAPv1.2] term meaning: 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.

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.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>system entity</td>
<td>An active element of a computer/network system. For example, an automated process or set of processes, a subsystem, a person or group of persons that incorporates a distinct set of functionality [SAMLGloss2].</td>
</tr>
<tr>
<td>user identifier</td>
<td>AKA user name or Principal.</td>
</tr>
<tr>
<td>web service</td>
<td>Generically, a service defined in terms of an XML-based protocol, often transported over SOAP, and/or a service whose instances, and possibly data objects managed therein, are concisely addressable via URIs. As specifically used in Liberty specifications, usually in terms of WSCs and WSPs, it means a web service that’s defined in terms of the ID-* &quot;stack&quot;, and thus utilizes [LibertySOAPBinding], [LibertySecMech], and is &quot;discoverable&quot; [LibertyDisco].</td>
</tr>
<tr>
<td>Web Service Consumer</td>
<td>A role donned by a system entity when it makes a request to a web service.</td>
</tr>
<tr>
<td>Web Service Provider</td>
<td>A role donned by a system entity when it provides a web service. Eve</td>
</tr>
</tbody>
</table>
4. Authentication Protocol

This section defines the ID-WSF Authentication Protocol. This protocol facilitates authentication between two ID-* entities, and is a profile of SASL [RFC2222].

4.1. Conceptual Model

The conceptual model for the ID-WSF Authentication Protocol is as follows: an ID-WSF system entity, acting in a Web Services Consumer (WSC) role, makes an authentication request to another ID-WSF system entity, acting in a Web Service Provider (WSP) role, and if the WSP is willing and able, an authentication exchange will ensue.

The authentication exchange is comprised of SOAP-bound ID-* messages [LibertySOAPBinding], and can involve an arbitrary number of round trips, dictated by the particular SASL mechanism employed [RFC2222]. The WSC may have out-of-band knowledge of the server’s supported SASL mechanisms, or it may send the server its own list of supported SASL mechanisms and allow the server to choose one from among them.

At the end of this exchange of messages, the WSC will either be authenticated or not, the nature of the authentication depending upon the SASL mechanism that was employed. Also depending on the SASL mechanism employed, the WSP may be authenticated as well.

Other particulars, such as how the WSC knows which WSP to contact for authentication, are addressed below in Section 6: Single Sign-On Service.

Note This document does not specify the use of SASL security layers.

4.2. Schema Declarations

The XML schema [Schema1] normatively defined in this section is constituted in the XML Schema file: liberty-idwsf-authn-svc-v2.0.xsd, entitled "Liberty ID-WSF Authentication Service XSD v2.0" (see Appendix C).

In addition, the Liberty ID-WSF Authentication Service XSD v2.0 explicitly includes, in the XML Schema sense, the Liberty ID-WSF Utility XSD v2.0 file (see Appendix D), whose filename is: liberty-idwsf-utility-v2.0.xsd, and the Liberty Utility XSD v2.0 file (see Appendix E), whose filename is: liberty-utility-v2.0.xsd.

4.3. SOAP Header Blocks and SOAP Binding

This specification does not define any SOAP header blocks. Section 4.3.1, below, constitutes the SOAP binding statement for this specification.

4.3.1. SOAP Binding

The messages defined below in Section 4.6, e.g. <SASLRequest>, are ordinary ID-* messages as defined in [LibertySOAPBinding]. They are intended to be bound to the [SOAPv1.1] protocol by mapping them directly into the <s:Body> element of the <s:Envelope> element comprising a SOAP message. [LibertySOAPBinding] normatively specifies this binding.
Implementations of this specification MUST use the `<sb:Correlation>` SOAP header block defined in [LibertySOAPBinding] to establish a message thread and thus correlate their authentication exchanges. See Section 5.5: Authentication Service Interaction Example for an example.

### 4.4. SASL Profile Particulars

The ID-WSF Authentication Protocol is based on SASL [RFC2222], and thus "profiles" SASL. Section 4 of [RFC2222] specifies SASL’s "profiling requirements". This section of this specification addresses some particulars of profiling SASL that are not otherwise addressed in the sections defining the protocol messages (Section 4.6: Protocol Messages), and their sequencing (Section 4.7: Sequencing of the Authentication Exchange).

#### 4.4.1. SASL "Service Name"

The SASL "Service Name" specified herein is: idwsf

#### 4.4.2. Composition of SASL Mechanism Names

The protocol messages defined below at times convey a SASL mechanism name, or a list of SASL mechanism names, as values of message element attributes.

These mechanism names are typically taken from the column labeled as "MECHANISMS" in [SASLReg], but MAY be site-specific.

These names, and lists of these names, MUST follow these rules:

- The character composition of a SASL mechanism name MUST be as defined in [IANA]’s SASL Mechanism Registry [SASLReg].
- A list of SASL mechanism names MUST be composed of names as defined above, separated by ASCII space chars (hex "20").

### 4.5. Authentication Exchange Security

This authentication protocol features the flexibility of having implementations being able to select at runtime the actual authentication mechanism (aka SASL mechanism) to employ. This however may introduce various vulnerabilities depending on the actual mechanism employed. Some mechanisms may be vulnerable to passive and/or active attacks. Also, since the server selects the SASL mechanism from a list supplied by the client, a compromised server, or a man-in-the-middle, can cause the weakest mechanism offered by the client to be employed.

Thus it is RECOMMENDED that the authentication protocol exchange defined herein (Section 4.7: Sequencing of the Authentication Exchange) be employed over a TLS/SSL channel [RFC2246] as amended by [RFC3546]. This will ensure the integrity and confidentiality of the authentication protocol messages. Additionally, clients SHOULD authenticate the server via TLS/SSL validation procedures. This will help guard against man-in-the-middle attacks.

### 4.6. Protocol Messages

This section defines the protocol’s messages, along with their message element attribute values, and their semantics. The sequencing of protocol interactions, also known as the authentication exchange, is defined below in Section 4.7: Sequencing of the Authentication Exchange.

#### 4.6.1. The `<SASLRequest>` Message

Figure 1 shows the schema fragment from Liberty ID-WSF Authentication Service XSD v2.0 describing the `<SASLRequest>` message. This message has the following attributes:
• **mechanism** [Required] — Used to convey a list of one-or-more client-supported SASL mechanism names to the server, or to signal the server if the client wishes to abort the exchange. It is included on all &lt;SASLRequest&gt; messages sent by the client.

• **authzID** [Optional] — The authzID, also known as user identifier or username or Principal, that the client wishes to establish as the "authorization identity" per [RFC2222].

• **advisoryAuthnID** [Optional] — The advisoryAuthnID may be used to advise the server what authentication identity will be asserted by the client via the selected SASL mechanism; i.e. it is a "hint". The advisoryAuthnID provides a means for server implementations to optimize their behavior on a per authentication identity basis.

  E.g. if a client requests to execute a certain SASL mechanism on behalf of some given authentication identity (represented by advisoryAuthnID) and authorization identity (represented by authzID) pair, the server can decide whether to proceed without having to execute the SASL mechanism (execution of which might involve more than a single round-trip). Server implementations that make use of the optional advisoryAuthnID attribute, SHOULD be capable of processing initial &lt;SASLRequest&gt; messages that do not include the advisoryAuthnID attribute.

• **id** [Optional] — identifies a &lt;SASLRequest&gt; message element 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.

```xml
<xs:element name="SASLRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Data" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:base64Binary"/>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="mechanism" type="xs:string" use="required"/>
    <xs:attribute name="authzID" type="xs:string" use="optional"/>
    <xs:attribute name="advisoryAuthnID" type="xs:string" use="optional"/>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
  </xs:complexType>
</xs:element>
```

Figure 1. &lt;SASLRequest&gt; Message Element — Schema Fragment

The &lt;SASLRequest&gt; message has the following sub-elements:
• <Data> — This element is used by the client to send SASL mechanism data to the server. In [RFC2222] parlance, this data is termed a "client response". Its content model is base64-encoded data.

• <samlp2:RequestedAuthnContext> — This element is used by the client to convey to the server a desired authentication context. It is used only on the initial SASL request (see Section 4.7: Sequencing of the Authentication Exchange). If present, the server uses the information in the <samlp2:RequestedAuthnContext> in combination with mechanism attribute when choosing the SASL mechanism to execute. The background use case for <samlp2:RequestedAuthnContext> is presented in Section 5.1: Authentication Service: Conceptual Model. See also: [LibertyAuthnContext] and [LibertyProtSchema].

Example 1. A SASLRequest Bound into a SOAP Message

4.6.1.1. <SASLRequest> Usage

The <SASLRequest> message is used to initially convey to the server a:

• list of one or more client-supported SASL mechanism names,

..in combination with optional:

• authzID attribute, and/or,

• advisoryAuthnID attribute, and/or,

• <samlp2:RequestedAuthnContext> element.
In the case where a single SASL mechanism name is conveyed, the `<SASLRequest>` message can contain a so-called initial response (see Section 5.1 of [RFC2222]) in the `<Data>` element.

If the server’s subsequent `<SASLResponse>` message signals that the authentication exchange should continue—and thus contains a server "challenge"—the client will send another `<SASLRequest>` message, with the `<Data>` element containing the client’s "response" to the challenge. This sequence of server challenges and client responses continues until the server signals a successful completion or aborts the exchange.

The mechanism attribute is used in these intermediate `<SASLRequest>` messages to signal the client’s intentions to the server. This is summarized in the next section.

Section 4.7: Sequencing of the Authentication Exchange, in combination with the next section, normatively defines the precise `<SASLRequest>` message format as a function of the sequencing of the authentication exchange.

### 4.6.1.2. Values for mechanism attribute of `<SASLRequest>`

The list below defines the allowable values for the mechanism attribute of the `<SASLRequest>` message element, and the resulting message semantics.

**Note**

In items #2 and #1, the mechanism attribute contains one or more SASL mechanism names, respectively. The rules noted in Section 4.4.2: Composition of SASL Mechanism Names MUST be adhered to in such cases.

1. **Multiple SASL mechanism names** — See Example 2. In this case, the `<SASLRequest>` message MUST NOT contain any "initial response" data, and MUST be the initial SASL request. See Section 4.6.2.1.2 for details on the returned `<SASLResponse>` message in this case.

   ```xml
   <SASLRequest mechanism="GSSAPI OTP PLAIN"/>
   ```

2. **A single SASL mechanism name** — In this case, the `<SASLRequest>` message MAY contain initial response data. See Example 3.

   ```xml
   <SASLRequest mechanism="GSSAPI">
   <Data>Q29ub3IgQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMgcGFzc3dvcmRzCg==</Data>
   </SASLRequest>
   ```

3. **A NULL string (""** — This indicates to the authentication server that the client wishes to abort the authentication exchange. See Example 4.

   ```xml
   <SASLRequest mechanism=""/>
   ```

**Example 4. `<SASLRequest>` Message Aborting the SASL Authentication Exchange**
4.6.2. The <SASLResponse> Message

Figure 2 shows the schema fragment from Liberty ID-WSF Authentication Service XSD v2.0 describing the <SASLResponse> message. This message has the following attributes:

- **serverMechanism** [Optional] — The server’s choice of SASL mechanism from among the list sent by the client.
- **id** [Optional] — identifies a <SASLResponse> message element 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.

```xml
<xs:element name="SASLResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Status"/>
      <xs:element ref="PasswordTransforms" minOccurs="0"/>
      <xs:element name="Data" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:base64Binary"/>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="serverMechanism" type="xs:string" use="optional"/>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
  </xs:complexType>
</xs:element>
```

Figure 2. <SASLResponse> Message Element - Schema Fragment

The <SASLResponse> message has the following sub-elements:

- **<Status>** — This element is from Liberty ID-WSF Utility XSD v2.0 and is used to convey status from the server to the client. See below.

- **<PasswordTransforms>** — This element is used to convey to the client any required password transformations. See Section 8: Password Transformations: The PasswordTransforms Element.

- **<Data>** — This element is used to return SASL mechanism data to the client. Its content model is base64-encoded data.
<wsa:EndpointReference> — This element is to convey to the client an ID-WSF EPR for the server, in its role as a WSP, upon a successful authentication exchange completion. Multiple instances of it may be used to also convey ID-WSF EPRs for additional instances of other services. Note that any credentials returned as a result of a successful authentication exchange are conveyed within any returned ID-WSF EPRs [LibertyDisco]. See Section 5: Authentication Service.

4.6.2.1. <SASLResponse> Usage

This message is sent by the server in response to a client <SASLRequest> message. It is used to convey "server challenges", in [RFC2222] parlance, to the client during an authentication exchange. So-called "client responses" are correspondingly conveyed to the server via the <SASLRequest> message, defined above. A given authentication exchange may occur in one "round-trip", or it may involve several round-trips. This depends on the SASL mechanism being executed.

The first <SASLResponse> sent by the server within an authentication exchange (as determined by the particular authentication mechanism being used) is explicitly distinguished from subsequent <SASLResponse> messages in terms of child elements and attributes. The final <SASLResponse> sent by the server in an authentication exchange is similarly distinguished, although with its own particular characteristics. These details are specified below in Section 4.7: Sequencing of the Authentication Exchange.

It is possible for different authentication mechanisms to be sequenced, the client authenticating to the server with one after another. For example, after a principal is authenticated with name and password (e.g. with PLAIN or CRAM-MD5), the service may (because of service or user policy) require additional authentication with SECUR-ID. Consequently, client implementations should be prepared for a message from the service with a "Continue" status code but a different "serviceMechanism" than that established in the previous authentication exchange. The message from the service that indicates such subsequent SASL mechanism may contain a <Data> element intended for processing by an implementation of the new mechanism. The client should process this message as specified in step 5 of Section 4.7: Sequencing of the Authentication Exchange.

The <Status> element (see Figure 3) is used to convey the authentication server’s assessment of the status of the authentication exchange to the client, via the code attribute (the <Status> element is declared in the Liberty ID-WSF Utility XSD v2.0).
In the two sections below, first the values of the `code` attribute of the `<Status>` element are discussed, followed by discussion of the various forms of `<SASLResponse>` messages and their semantics.

### 4.6.2.1.1. Values for the `code` attribute of `<Status>`

If the value of `code` is:

- "Continue" — the server expects the client to craft and send a new `<SASLRequest>` message containing data appropriate for whichever step the execution of the SASL mechanism is at.
- "OK" — the server considers the authentication exchange to have completed successfully.

The `<SASLResponse>` message will typically contain ID-WSF EPR(s) (i.e. `<wsa:EndpointReference>` element(s)) containing credentials, as described below in Section 5.3: Rules for Authentication Service Providers, enabling the client to interact further with this provider, for example to invoke another ID-WSF service such as the Discovery Service.

Additionally, the `<SASLResponse>` message can convey ID-WSF EPRs for other providers.

See Section 4.7: Sequencing of the Authentication Exchange for the normative specification of the composition of the `<SASLResponse>` message in this case. See also Section 5.3: Rules for Authentication Service Providers.

- "Abort" — the server is aborting the authentication exchange. It will not send any more messages on this message thread.

### 4.6.2.1.2. Returning the Server’s Selected SASL Mechanism

The server will choose one SASL mechanism from among the intersection of the list sent by the client and the server’s set of supported and willing-to-execute SASL mechanisms. It will return the name of this selected SASL mechanism as the value for the `serverMechanism` attribute on the initial `<SASLResponse>` message. See Example 5.

```
<SASLResponse serverMechanism="DIGEST-MD5">
  <Status code="Continue"/>
  <Data>
    Q29ub3IgQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMvcGFzc3dvcmRzCg==
  </Data>
</SASLResponse>
```

Example 5. `<SASLResponse>` Indicating Server’s Chosen SASL Mechanism

If there is no intersection between the client-supplied list of SASL mechanisms and the set of supported, and willing-to-execute, server-side SASL mechanisms, then the server will return a `<SASLResponse>` message with a `code` attribute whose value is "Abort". See Example 6, and also item #3 in Section 4.7: Sequencing of the Authentication Exchange.

```
<SASLResponse>
  <Status code="Abort"/>
</SASLResponse>
```

Example 6. `<SASLResponse>` Indicating a Server-side Abort

### 4.7. Sequencing of the Authentication Exchange

The authentication exchange is sequenced as follows:
1. The authentication exchange MUST begin by the client sending the server a `<SASLRequest>` message. This message:

   • MUST contain a `mechanism` attribute whose value is a string containing one or more SASL mechanisms the client supports and is prepared to negotiate (see Section 4.6.1.2: Values for `mechanism` attribute of `<SASLRequest>`).

   • MAY contain a `<Data>` element containing an initial response, specific to the cited SASL mechanism, if the `mechanism` attribute contains only a single SASL mechanism. See section 5.1 of [RFC2222].

   • MAY contain a `<samlp2:RequestedAuthnContext>` element.

   • SHOULD contain an `authzID` attribute whose value is an identifier string for the Principal being authenticated.

   • MAY contain an `advisoryAuthnID` attribute whose value is an identifier asserted by the client to represent the authentication identity being established by this authentication event.

   • MAY contain an `id` attribute.

2. If the server is prepared to execute, with this client, at least one of the SASL mechanism(s) cited by the client in the previous step, then processing continues with step 4.

3. Otherwise, the server does not support, or is not prepared to negotiate, any of the SASL mechanisms cited by the client. The server MUST respond to the client with a `<SASLResponse>` message containing:

   • A `<Status>` element with a `code` attribute with a value of "Abort".

   • No `<PasswordTransforms>` element.

   • No `<Data>` element.

   • No `<wsa:EndpointReference>` element.

   • No `serverMechanism` attribute.

   The `<SASLResponse>` message MAY have an `id` attribute. After this message is sent to the client, processing continues with step 7.

4. The server sends to the client a `<SASLResponse>` message.

   If this message is the first `<SASLResponse>` sent to the client in this authentication exchange (as determined by a particular authentication mechanism), this message:

   • MUST contain a `serverMechanism` attribute whose value is a single SASL mechanism name, chosen by the server from the list sent by the client.

   • MAY contain a `<Data>` element containing a SASL mechanism-specific challenge.

   • MAY contain a `<PasswordTransforms>` element. See Section 8: Password Transformations: The `PasswordTransforms` Element for details on the client’s subsequent obligations in this case.

   • MAY contain a `<id>` attribute.

   • MUST contain a `<Status>` element with a `code` attribute whose value is given by either item A, or B, or C:
A. "Continue" — either the execution of the SASL mechanism is not complete or the authentication exchange was successful but the server expects the client to authenticate again using a different authentication mechanism; the server expects the client to process this message and respond.

If the server is indicating that the client should continue by authenticating with a different mechanism, the server MUST specify the desired mechanism as the value for "serverMechanism". The authentication mechanism specified MUST be taken from the list previously sent by the client in the prior authentication exchange. The server MAY include a <Data> element (and <PasswordTransforms>) with content appropriate for the new authentication mechanism.

If the reason for the server indicating that the client should continue is that the client presented invalid credentials, the server SHOULD include a second level status <Status code="InvalidCredentials">. The server MAY also return a <Data> element (e.g. with a new challenge according to the mechanism already established) and the client can respond according to the mechanism. Processing continues with step 5.

B. "OK" — the server declares the authentication exchange has completed successfully.

In this case, this final SASL response message can contain, in addition to the items listed above, <wsa:EndpointReference> element(s), containing requisite credentials. This is specified in Section 5.3: Rules for Authentication Service Providers.

Processing continues with step 6.

C. "Abort" — the server declares the authentication exchange has completed unsuccessfully. For example, the user may have supplied incorrect information, such as an incorrect password. See step 7, below, for additional information.

In this case, this <SASLResponse> message MUST NOT contain any <wsa:EndpointReference> element(s).

Processing continues with step 7.

Otherwise, this message:

- MUST NOT contain a serverMechanism attribute.
- MAY contain a <Data> element containing a SASL mechanism-specific challenge.
- MUST NOT contain a <PasswordTransforms> element.
- MAY contain a <id> attribute.
- MUST contain a <Status> element with a code attribute whose value is given by either item A, or B, or C:

A. "Continue" — the execution of the SASL mechanism is not complete; the server expects the client to process this message and respond. Processing continues with step 5.

B. "OK" — the server declares the authentication exchange has completed successfully.

In this case, this "final response" <SASLResponse> message can contain, in addition to the items listed above, <wsa:EndpointReference> element(s) with requisite credentials. This is specified in Section 5.3: Rules for Authentication Service Providers.

Processing continues with step 6.
C. "Abort" — the server declares the authentication process has completed unsuccessfully. For example, the user may have supplied incorrect information, such as an incorrect password. If the reason for the server aborting is that the client presented invalid credentials, the server SHOULD include a second level status <Status code="InvalidCredentials">. In this case, this <SASLResponse> message MUST NOT contain any <wsa:EndpointReference> element(s).

Processing continues with step 7.

5. The client sends the server a <SASLRequest> message. This message:

- SHOULD contain a mechanism attribute set to the same value as sent by the server, as the value of the serverMechanism attribute, in its first <SASLResponse> message (see Section 4.6.2.1.2: Returning the Server’s Selected SASL Mechanism).

Note

The client MAY, however, choose to abort the authentication exchange by setting the mechanism attribute to either a "null" string, or to a mechanism name different than the one returned by the server in its first <SASLResponse> message.

If the client chooses to abort, processing continues with step 8.

- SHOULD contain a <Data> element containing data specific to the cited SASL mechanism.

- MUST NOT contain a <samlp2:RequestedAuthnContext> element.

- MAY contain an id attribute.

Processing continues with steps 4 and 5 until the server signals success, failure, or aborts — or the client aborts the exchange using the technique noted in the first bullet item, above, of this step.

6. The authentication exchange has completed successfully. The client is now authenticated in the server’s view, and the server may be authenticated in the client’s view, depending upon the SASL mechanism employed. Section 5.1: Authentication Service: Conceptual Model discusses what the next interaction steps between the client and server are in the ID-WSF authentication service case.

7. The authentication exchange has completed unsuccessfully due to an exception on the server side. The client SHOULD cease sending messages on this message thread. The reasons for an authentication exchange failing are manifold. Often it is simply a case of the user having supplied incorrect information, such as a password or passphrase. Or, there may have been a problem on the server’s part, such as an authentication database being unavailable or unreachable.

Note

[RFC2222] and the RFCs specifying various SASL mechanisms—for example [RFC2245], [RFC2444], and [RFC3163]—are arguably not as clear as they could be with respect to the situation where an execution of the SASL mechanism fails for some reason. Though, Section 4, item 3, of [RFC2222] indicates that the server must have a means of indicating "failure of the exchange" to the client. In this version of this specification, this is handled by the server returning a status code of "Abort" to the client, as specified above in 4. Future versions of this specification may facilitate more fine-grained error reporting by the server.

8. The client aborted the authentication exchange.
5. Authentication Service

The ID-WSF Authentication Service provides web service-based authentication facilities to Web Service Consumers (WSCs). This service is built around the SASL-based ID-WSF Authentication Protocol as specified above in Section 4. This section first outlines the Authentication Service’s conceptual model and then defines the service itself.

5.1. Conceptual Model

ID-WSF-based Web Service Providers (WSPs) may require requesters, AKA Web Service Consumers (WSCs), to present security tokens in order to successfully interact (security token specifics, are specified in [LibertySecMech]).

A Discovery Service [LibertyDisco], which itself is just a WSP, is able to create security tokens authorizing WSCs to interact with other WSPs, on whose behalf a Discovery Service has been configured to speak. Also, Discovery Service instances might themselves be configured to require WSCs to present security tokens when making requests of them.

The ID-WSF Authentication Service addresses the above conundrum by providing the means for WSCs to prove their identities—to authenticate—and obtain security tokens enabling further interactions with other services, at the same provider, on whose behalf the Authentication Service instance is authorized to speak. These offered services may be, for example, a Discovery Service or Single Sign-On Service. WSCs may then use these latter services to discover and become capable of interacting with yet other services.

Note that although an Authentication Service itself does not require requesters to present security tokens in order to interact with it, an Authentication Service may, in some situations, be configured to understand presented security tokens and use them when applying policy.

5.1.1. Stipulating a Particular Authentication Context

In some situations, a WSC may need to stipulate some of the properties for an authentication exchange. A scenario illustrating a use case of this is:

Suppose a Principal is wielding a Liberty-enabled user agent or device (LUAD) that is acting as a WSC (i.e. a LUAD-WSC). The Principal authenticates with her bank, say, and authenticates via the ID-WSF authentication service using some authentication mechanism, such as PLAIN [SASLReg]. At some point, the Principal wants to transfer a large sum of money to the Fund for Poor Specification Editors (using some (fictitious) ID-SIS-based web service), and the bank’s system indicates to the LUAD-WSC that the Principal’s present authentication is “inappropriate”. The bank’s system also includes a <RequestedAuthnContext>.

Now, the LUAD-WSC “knows” that it needs to help the Principal reauthenticate—as her present credentials aren’t being honored for the financial transaction she wishes to carry out. So the LUAD-WSC prompts the Principal for permission to reauthenticate her, and (assuming the answer was “yes”) initiates the ID-WSF Authentication Protocol with the appropriate authentication service provider, and includes the supplied-by-the-bank <RequestedAuthnContext>. The authentication service provider factors the requested authentication context into its selection of SASL mechanism for the ensuing authentication exchange. And upon successful authentication, the Principal is able to successfully make the funds transfer.

When initiating an authentication exchange, a WSC can stipulate some properties for the ensuing authentication event, and thus the subsequently issued (if successful) credentials. It does this by including a <RequestedAuthnContext> in the initial <SASLRequest>.

5.2. Service Type Declaration

The Service Type URI for the ID-WSF Authentication Service is:

urn:liberty:as:2004-04
Note

The above Service Type URI, having as subcomponents "liberty:as", is explicitly distinct from the Liberty Authentication Protocol and Service URI defined in Section 2.2: XML Namespaces. The latter has as subcomponents "liberty:sa"—representing "soap authentication" (a historical and incorrect appellation)—thus occasionally causing confusion.

5.3. Rules for Authentication Service Providers

Providers offering ID-WSF Authentication Services MUST adhere to the following rules:

1. Authentication Service Providers (AS Providers) MUST implement the ID-WSF Authentication Protocol, as defined in Section 4: Authentication Protocol. The Authentication Service Provider MUST play the role of the authentication server.

2. Upon successful completion of an authentication exchange the first ID-WSF EPR, as materialized as an <wsa:EndpointReference> element instance and contained in the final SASL response, SHOULD refer to services at the Authentication Service provider—i.e. at the "same provider"—that said AS Provider can offer to the Authentication Service consumer.

   For example, Identity Providers may often also include an ID-WSF EPR for the Discovery Service of the Principal just authenticated, as well as ID-WSF EPRs for other offered services, such as an SSO Service.

   See Section 4.7: Sequencing of the Authentication Exchange, Step 4.

   The Provider MAY also include additional ID-WSF EPRs referring to services offered by other providers—i.e. providers other than the AS Provider.

3. Any included credentials SHOULD be useful for a reasonable time (note that credentials will be contained within the ID-WSF EPRs, as profiled in [LibertyDisco]). Even if the AS Consumer recently authenticated with the Authentication Service, i.e. an earlier issued credential for consumption by the AS Provider is still valid, the AS Provider SHOULD issue credential(s) that have later expiration times than the earlier issued credential(s).

   The AS Provider MAY choose to re-authenticate, using any of the available SASL mechanisms, or issue new credentials without engaging in an authentication exchange. This can be accomplished by responding to the AS Consumer’s initial SASL request with a final SASL response containing an ID-WSF EPR, itself containing the requisite credentials.

   Note

   Credentials containing <saml:AuthenticationStatement>(s) (SAML v1.1) or <saml2:AuthnStatement>(s) (SAML v2.0) should have their <saml:AuthenticationInstant>(s) (SAML 1.1) or <saml2:AuthnInstant>(s) set to the time when the authentication event actually took place. See [SAMLCore11] and/or [SAMLCore2] as appropriate.

4. Additionally, if the first <SASLRequest> in an exchange contains a <samlp2:RequestedAuthnContext> element, then upon successful authentication, the Authentication Service MUST either: return credentials (embedded within returned ID-WSF EPR(s)) that satisfy the <lib:RequestedAuthnContext>, or, abort the authentication exchange (see also the "Single Sign-On and Federation Protocol" section in [LibertyProtSchema]).

   To satisfy the <lib:RequestedAuthnContext>, any returned credentials MUST be created according to the following rules:

   a. If one or more <saml2:AuthnContextClassRef> or <saml2:AuthnContextDeclRef> elements are present in the <lib:RequestedAuthnContext>, then the resulting authentication statement in the assertion (if any) MUST contain an authentication statement that conforms to the class or statement specified.

   Additionally, the set of supplied elements MUST be evaluated as an ordered set, where the first element is the most preferred authentication context class or statement. If none of the specified classes or statements can be satisfied, the identity provider MUST NOT include a credential and abort.
b. Additionally, if a Comparison attribute is supplied, and one or more <saml2:AuthnContextClassRef>
or <saml2:AuthnContextDeclRef> elements are included, then the resulting authentication statement
in the assertion (if any) MUST follow the rule specified in the Comparison attribute. If this requirement
cannot be satisfied, the identity provider MUST NOT include a credential and abort.

c. If Comparison is specified and set to "exact", then the resulting authentication statement in the assertion (if any) MUST be the exact match of at least one of the authentication contexts specified.

If Comparison is specified and set to "minimum", then the resulting authentication statement in the assertion (if any) MUST be at least as strong (as deemed by the Authentication Service provider) as one of the
authentication contexts specified.

If Comparison is specified and set to "better", then the resulting authentication statement in the assertion (if any) MUST be stronger (as deemed by the identity provider) than any specified in the supplied authentication contexts.

If Comparison is specified and set to "maximum", then the resulting authentication statement in the assertion (if any) MUST be as strong as possible (as deemed by the identity provider) without exceeding the
strength of at least one of the authentication contexts specified.

5. An Authentication Service instance SHOULD be deployed such that the security mechanism [LibertySecMech]:

urn:liberty:security:2003-08:TLS:null

can be used by the WSC.

Note

In practice this means that the Authentication Service should be exposed on an endpoint for which the URL
should have https as the protocol field.

6. An Authentication Service implementation SHOULD support the following SASL mechanisms [SASLReg]:

PLAIN, CRAM-MD5.

5.4. Rules for Authentication Service Consumers

WSCs implementing the client-side of the ID-WSF Authentication Protocol, and thus also known as Authentication
Service Consumers (AS Consumers), MUST adhere to the following rules:

1. AS Consumers MUST implement the ID-WSF Authentication Protocol, as defined in Section 4: Authentication
Protocol in the role of the client.

Note

The AS Consumer may include various SOAP header blocks, e.g. a <wsse:Security> element [Liberty-
SecMech] which can house a security token(s) obtained earlier from an Authentication Service or Discovery
Service [LibertyDisco]. In such a case, the Authentication Service SHOULD evaluate the presented security to-
ken(s) in combination with applicable policy, as a part of the overall authentication event. This provides a means,
for example, of "security token renewal".

2. In case the AS Consumer has not been provisioned with the <disco:SecurityMechID> for the Authentication
Service instance that it uses, the AS Consumer SHOULD assume that the required security mechanism is this
one:

urn:liberty:security:2003-08:TLS:null
Note

<disco:SecurityMechID> elements are contained within the <disco:SecurityContext> element(s), themselves occurring within ID-WSF EPRs (profiled <wsa:EndpointReference>s) [LibertyDisco]).

Only when the endpoint URL of the Authentication Service is prescribed to have http as the protocol MAY the WSC presume a security mechanism of:

urn:liberty:security:2003-08:null:null

3. It is RECOMMENDED that the WSC support the password transformations specified in Appendix B.

5.5. Authentication Service Interaction Example

Example 7 through Example 10 illustrate an example exchange between a LUAD-WSC and an ID-WSF Authentication Service (AS). The AS includes information about the Discovery Service (DS) in its final response. Here the DS is offered by the same provider.

Example 7. The WSC sends a <SASLRequest> on behalf of a Principal, asserting that the authentication identity is "358408021451" and indicates it desire to use the "CRAM-MD5" SASL mechanism.

Example 8. The AS replies, agreeing to use CRAM-MD5, and issues a CRAM-MD5 challenge.
Example 9. The WSC responds with an CRAM-MD5 response.
Example 10. The AS replies with its "final" <SASLResponse> message, which includes credentials with which the WSC may subsequently use to invoke a DS.
6. Single Sign-On Service

The ID-WSF Single Sign-On Service (SSO Service, or SSOS) provides Web Service Consumers (WSCs) an ID-WSF-based means to obtain Liberty authentication assertions enabling them to interact with Service Providers (SPs) [LibertyProtSchema].

This section first outlines the ID-WSF SSO Service’s conceptual model and then defines the SSO Service in terms of rules for Providers and Consumers of the service.

6.1. Conceptual Model

In the Liberty architecture, it is conceivable for any concrete system entity to don any architectural role that it is physically capable of bearing. For example, a Liberty Service Provider (SP) is essentially just a Liberty ID-FF-enabled website. Such Service Providers can also be simultaneously cast into WSC and WSP roles.

Similarly, user agents in the Liberty architecture range from vanilla web browsers, to modestly Liberty-enabled browsers (LECPs), to arbitrarily complex SOAP-based clients. These latter user agents, termed Liberty-enabled User Agents or Devices (LUADs) will conceivably be dynamically cast into the full range of Liberty architectural roles; they will be called upon to be a vanilla browser one moment, and a WSC the next, and even a WSP at times.

Similarly to the conundrum outlined in Section 5: Authentication Service, a LUAD acting as a WSC one moment (a "LUAD-WSC") and as a vanilla browser the next, will need the means to obtain authentication assertions and security tokens as necessary.

As noted in Section 5, a (LUAD-)WSCs needing to obtain security tokens in order to interact with a Discovery Service can utilize an ID-WSF Authentication Service to obtain requisite security tokens. However, in ID-FF, the user agent is assumed to be a vanilla browser, and Identity Providers vouch for browser-wielding Principals by sending authentication assertions, or "pointers" to authentication assertions (AKA "SAML artifacts" [SAMLCore11]), to Service Providers "through" Principals’ browsers (e.g. via HTTP "redirects").

(LUAD-)WSCs thus need some means to cause authentication assertions to be conveyed to SPs they wish to interact with. Remember that not all SPs will be able to don a WSP role, so simply authenticating via the Authentication Service, either at some Identity Provider or with an SP/WSP providing an Authentication Service, is not a solution for this use case.

The ID-WSF Single Sign-On Service addresses this issue. It is a profile of the ID-FF Single Sign-On and Federation Protocol [LibertyProtSchema]. It provides the means for (LUAD-)WSCs to interact with SPs. See also [LibertyClient-Profiles] for additional background information.

The overall mechanism is based on two steps. First, a (LUAD-)WSC wishing to interact with some SP can use the Authentication Service at an Identity Provider to obtain security tokens. Next, the (LUAD-)WSC invokes the Single Sign-On Service at the Identity Provider in order to obtain an authentication assertion to convey to the SP, thus enabling Liberty-SSO-enabled, vanilla, web-based interactions with that SP.

For example, if a (LUAD-)WSC successfully authenticates with an Identity Provider (IdP) via the IdP’s Authentication Service (Section 5), the IdP can ensure that the LUAD-WSC will have in its possession an ID-WSF EPR (a profiled <wsa:EndpointReference>; [LibertyDisco]), containing any necessary credentials, for the ID-WSF Single Sign-On Service at the very same IdP. Thus the (LUAD-)WSC may obtain an authentication assertion via the IdP’s the latter Service.

Additionally, the IdP can, at the same time, ensure that the (LUAD-)WSC possesses an ID-WSF EPR, containing any necessary credentials, for the Discovery Service (DS) of the Principal wielding the LUAD-WSC — thus enabling the LUAD-WSC to simultaneously utilize ID-FF- and ID-WSF-based services on behalf of the Principal, based on one sign-on interaction, from the Principal’s perspective.
In yet another plausible scenario, some web service provider(s) might not be ID-WSF-based. Rather, they could be *generic Web Service Providers* (gWSPs).

A (LUAD-)WSC consuming services from gWSPs may need to obtain security tokens satisfying whichever security paradigm the gWSPs employ. It is plausible that such a paradigm will accommodate use of Liberty authentication assertions as security tokens — for example, see [wss-sms] and [wss-saml]. Note that the SSO Service can address this use case.

### 6.2. Service Type Declaration

The Service Type URI for the ID-WSF SSO Service is:

```
urn:liberty:ssos:2004-04
```

### 6.3. Rules for SSO Service Providers

SSO Service Providers (SSOS Providers) MUST adhere to the following rules:

1. Unless stated otherwise below the SSOS Provider SHOULD adhere to the [SAMLCore2] and [SAMLBind2] specifications.

2. The SSOS Provider SHOULD offer an ID-WSF Authentication Service, as defined in Section 5: Authentication Service. Upon successful authentication the SSOS Provider will respond to the SSOS Consumer with a `<SASLResponse>` message, as specified in Section 5. Returned ID-WSF EPRs referring to SSO Service instances MUST use the Service Type URI defined in Section 6.2 above.

3. The SSOS Provider SHOULD adhere to the SOAP binding as specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the [LibertySOAPBinding] shall take precedence.

4. SSOS Providers SHOULD advertise their SSO capability in metadata [LibertyMetadata]. To accomplish this, it MUST include a `<md:SingleSignOnProtocolProfile>` element in its metadata, with a value of:

```
urn:liberty:iif:profiles:id-wsf
```

5. The SSOS Provider MAY, when it receives an `<samlp2:AuthnRequest>` that has its `ProtocolBinding` element set to `urn:liberty:iif:profiles:id-wsf`, respond with an ID-WSF message, containing relevant header blocks as specified in [LibertySOAPBinding].

*Note*

SSOS Providers MAY take advantage of various optional header blocks defined in [LibertySOAPBinding]. For example, instead of attempting to establish a local session via an HTTP cookie, which is likely to be ignored, the SSOS Provider may include a `<sec:ServiceSessionContext>` element in a `<sb:ServiceInstanceUpdate>` header block. The WSC that sent the original `<samlp2:AuthnRequest>` must of course understand these header blocks.

6. SSOS Providers SHOULD NOT respond with any content other than SOAP. For example, the MIME type of the HTTP response must be set according to [LibertySOAPBinding]
Note

This is different from the LECP profile [SAMLBind2] where an IdP is allowed to respond with any content that is acceptable to the requester (i.e. the LECP).

7. Upon successful processing of the <samlp2:AuthnRequest>, the SSOS Provider SHOULD respond with a SOAP-bound <samlp2:Response> message, constructed according to [SAMLCore2] in combination with [SAMLBind2].

Note

This is different from the LECP profile [SAMLBind2] where an IdP is expected to respond with an <samlp2:Response>.

6.4. Rules for SSO Service Consumers

Consumers of the ID-WSF SSO Service MUST adhere to the following rules:

1. Unless stated otherwise below the WSC SHOULD adhere to the rules for active intermediaries as specified in [SAMLCore2] and [SAMLBind2].

2. The WSC SHOULD adhere to the SOAP binding as specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the [LibertySOAPBinding] shall take precedence. For example, the WSC must include a proper <sb:Correlation> header block in its messages to the SSOS Provider.

Note

The WSC MAY include various other header blocks, e.g. a <wsse:Security> header block [LibertySecMech] [wss-sms]. Such a header block could contain a security token obtained from an ID-WSF Authentication Service Provider.

3. The WSC MUST set the ProtocolBinding attribute of the <samlp2:AuthnRequest> to:

   urn:liberty:iff:profiles:id-wsf

Note

Depending on the application and deployment model the WSC may have to construct the <samlp2:AuthnRequest> by itself, unlike LECP implementations that merely repack a message that was constructed by an SP. Obviously, a WSC will not be able to sign <samlp2:AuthnRequest> messages on behalf of the party that will consume the <samlp2:Response>. See the discussion in Section 6.1: Conceptual Model for context.

4. When the WSC receives security tokens, in the form of <saml2:Assertion> elements or derivatives thereof, it MUST NOT send these to any other party than the intended audience, as indicated in the assertion’s <saml2:Audience> element.
7. Identity Token Mapping Service

This section describes the Identity Token Mapping Service. "identity tokens" are defined in [LibertySecMech].

Using the request messages illustrated in Figure 4, a requester may obtain, exchange (aka "map"), update, and validate identity tokens.

An example employer of this service is the People Service [LibertyPeopleService].

```xml
<xs:element name="TokenRequest">
    <xs:complexType>
        <xs:sequence>
            <xs:choice>
                <xs:element ref="samlp:NameID"/>
                <xs:element ref="samlp:EncryptedID"/>
            </xs:choice>
            <xs:element ref="samlp:NameIDPolicy"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="TokenMap">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="sec:Token" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="samlp:NameIDPolicy"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="TokenUpdate">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="sec:Token" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="TokenValidate">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="sec:Token" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="TokenResponse">
    <xs:complexType>
        <xs:sequence>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```
Figure 4. Identity Token Protocols Schema

[TODO: complete prose for above, add examples]
8. Password Transformations: The PasswordTransforms Element

This section defines the `<PasswordTransforms>` element. Authentication servers MAY use this element to convey password pre-processing obligations to clients.

For example, an authentication server may have been configured such that it presumes that the strings users enter as their passwords have been pre-processed in some fashion before being further processed and/or stored. For example, the passwords may be truncated to a given length, and all upper case characters may be folded to lower case, and whitespace may be eliminated. The authentication server can communicate these requirements dynamically to clients using the `<PasswordTransforms>` element in an initial `<SASLResponse>`. See Figure 5.

```xml
<xsd:element name="PasswordTransforms">
  <xsd:annotation>
    <xsd:documentation>
      Contains ordered list of sequential password transformations
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Transform" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:complexContent>
            <xsd:extension base="xsd:string">
              <xsd:attribute name="name" type="xsd:string" use="required"/>
            </xsd:extension>
          </xsd:complexContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 5. The PasswordTransforms element
<PasswordTransforms>
  <Transform name="urn:liberty:sa:pm:truncate">
    <Parameter name="length">8</Parameter>
  </Transform>
  <Transform name="urn:liberty:sa:pm:lowercase"/>
</PasswordTransforms>

Figure 6. Example of a PasswordTransforms

Servers MAY include a <PasswordTransforms> element along with their initial <SASLResponse> to a client. A <PasswordTransforms> element contains one or more <Transform> elements. Each <Transform> is identified by the value of the name attribute which must be a URI [RFC2396]. This URI MUST specify a particular transformation on the password. Transforms are specified elsewhere, for example in configuration data at implementation- and/or deployment-time. A basic set is specified in Appendix B: Password Transformations.

A client receiving an initial <SASLResponse> message containing a <PasswordTransforms> element MUST apply the specified transformations to any password that is used as input for the SASL mechanism indicated in the <SASLResponse>.

The client MUST apply the transformations in the order given in the <PasswordTransforms> element, and MUST apply each transform to the result of the preceding transform. Of course, the first transform MUST be applied to the raw password.

Unless the specification of a <Transform> states otherwise, it is specified in terms of [Unicode] abstract characters. An abstract character is a character as rendered to a user. Since an abstract character may require more than one octet to represent, there is not necessarily a one-to-one mapping between an abstract character, or sequence of abstract characters, and its corresponding coded character representation.

For example, if a truncation transform indicates, "truncate after the first eight characters", the characters after the eighth abstract character should be removed; in some languages and character encodings this could mean that more than 8 octets remain.

See also Appendix B.
9. Acknowledgments

This spec leverages techniques and ideas from draft-nystrom-http-sasl-xx (an IETF Internet-Draft), RFC3080, RFC2251, RFC2829, RFC2830, et al (all are various IETF Requests For Comments). The authors of those specs are gratefully acknowledged. Thanks also to Alexy Melnikov, Paul Madsen, Scott Cantor, and RL "Bob" Morgan for their feedback and insights. The docbook source code for this specification was hand set to the tunes of Brad, Bob Mould, Weather Report, Miles Davis, John Coltrane, Liz Phair, The Wallflowers, Alan Holdsworth, Chick Corea, Jennifer Trynin, Elisa Korenne, The Cowboy Junkies, Fugazi, Blues Traveler, Blink-182, CSN, Pearl Jam, and various others. Thanks also to whatever deities are responsible for the existence of coffee, dark chocolate, and fermented cereals.
References

Normative


Informational


### A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms

Ref: [SASLReg]

Note

The file listed below IS SUBJECT TO CHANGE! It is presented here as non-normative background information only. Implementers and deployers should always retrieve the a fresh copy of this file from [IANA].

**SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS**

(last updated 2004-01-21)

SASL mechanisms are named by strings, from 1 to 20 characters in length, consisting of upper-case letters, digits, hyphens, and/or underscores. SASL mechanism names must be registered with the IANA. Procedures for registering new SASL mechanisms are given in the section "Registration procedures" of RFC2222.

<table>
<thead>
<tr>
<th>MECHANISMS</th>
<th>USAGE</th>
<th>REFERENCE</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>KERBEROS_V4</td>
<td>LIMITED</td>
<td>[RFC2222]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>GSSAPI</td>
<td>COMMON</td>
<td>[RFC2222]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>SKEY</td>
<td>OBSOLETE</td>
<td>[RFC2444]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>COMMON</td>
<td>[RFC2222]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>CRAM-MD5</td>
<td>LIMITED</td>
<td>[RFC2195]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>ANONYMOUS</td>
<td>COMMON</td>
<td>[RFC2245]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>OTP</td>
<td>COMMON</td>
<td>[RFC2444]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>GSS-SPNEGO</td>
<td>LIMITED</td>
<td>[Leach]</td>
<td>Paul Leach <a href="mailto:paulle@microsoft.com">paulle@microsoft.com</a></td>
</tr>
<tr>
<td>PLAIN</td>
<td>COMMON</td>
<td>[RFC2595]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>SECURID</td>
<td>COMMON</td>
<td>[RFC2808]</td>
<td>Magnus Nyström <a href="mailto:magnus@rsasecurity.com">magnus@rsasecurity.com</a></td>
</tr>
<tr>
<td>NTLM</td>
<td>LIMITED</td>
<td>[Leach]</td>
<td>Paul Leach <a href="mailto:paulle@microsoft.com">paulle@microsoft.com</a></td>
</tr>
<tr>
<td>NMAS_LOGIN</td>
<td>LIMITED</td>
<td>[Gayman]</td>
<td>Mark G. Gayman <a href="mailto:mgayman@novell.com">mgayman@novell.com</a></td>
</tr>
<tr>
<td>NMAS_AUTHEN</td>
<td>LIMITED</td>
<td>[Gayman]</td>
<td>Mark G. Gayman <a href="mailto:mgayman@novell.com">mgayman@novell.com</a></td>
</tr>
<tr>
<td>DIGEST-MD5</td>
<td>COMMON</td>
<td>[RFC2831]</td>
<td>IESG <a href="mailto:iesg@ietf.org">iesg@ietf.org</a></td>
</tr>
<tr>
<td>9798-U-RSA-SHA1-ENC</td>
<td>COMMON</td>
<td>[RFC3163]</td>
<td><a href="mailto:robert.zuccherato@entrust.com">robert.zuccherato@entrust.com</a></td>
</tr>
<tr>
<td>9798-M-RSA-SHA1-ENC</td>
<td>COMMON</td>
<td>[RFC3163]</td>
<td><a href="mailto:robert.zuccherato@entrust.com">robert.zuccherato@entrust.com</a></td>
</tr>
<tr>
<td>9798-U-DSA-SHA1</td>
<td>COMMON</td>
<td>[RFC3163]</td>
<td><a href="mailto:robert.zuccherato@entrust.com">robert.zuccherato@entrust.com</a></td>
</tr>
</tbody>
</table>
9798-M-DSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
9798-U-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
9798-M-ECDSA-SHA1 COMMON [RFC3163] robert.zuccherato@entrust.com
KERBEROS_V5 COMMON [Josefsson] Simon Josefsson <simon@josefsson.org>

References
----------

People
------
[Leach] Paul Leach, <paulle@microsoft.com>, December 1998, June 2000.
B. Password Transformations

This section defines a number of password transformations.

1. Truncation

The urn:liberty:sa:pw:truncate transformation instructs processors to remove all (Unicode abstract) subsequent characters after a given number of characters have been obtained (from the user). Subsequent processing MUST take only the given number of characters as input. The number of characters that shall remain is given in a <Parameter> element with name "length".

```
<Transform name="urn:liberty:sa:pw:truncate">
  <Parameter name="length">8</Parameter>
</Transform>
```

Figure B.1. Example of truncation transformation

2. Lowercase

The urn:liberty:sa:pw:lowercase transformation instructs processors to replace all uppercase characters with lowercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters. Note that the "case" of the abstract Unicode character is decisive, i.e. only characters that have the Uppercase property should be replaced with equivalent characters with the Lowercase property. This mapping from UPPERCASE to lowercase should confirm to the relevant sections (e.g. 4.2) of [Unicode].

```
<Transform name="urn:liberty:sa:pw:lowercase" />
```

Figure B.2. Example of lowercase transformation

3. Uppercase

The urn:liberty:sa:pw:uppercase transformation instructs processors to replace all lowercase characters with uppercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters. Note that the "case" of the abstract Unicode character is decisive, i.e. only characters that have the Lowercase property should be replaced with equivalent characters with the Uppercase property. This mapping from lowercase to UPPERCASE should confirm to the relevant sections (e.g. 4.2) of [Unicode].

```
<Transform name="urn:liberty:sa:pw:uppercase" />
```

Figure B.3. Example of uppercase transformation

4. Select

The urn:liberty:sa:pw:select transformation instructs processors to remove all characters except those specified in the "allowed" parameter. Note that the allowed characters refer to abstract Unicode characters. In the message that contains the <Transform> element these characters are encoded with the same encoding as used for the xml document that contains the message (usually UTF-8).
<Transform name="urn:liberty:sa:pw:select">
  <Parameter name="allowed">0123456789abcdefghijklmnopqrstuvwxyz</Parameter>
</Transform>

Figure B.4. Example of select transformation
C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
targetNamespace="urn:liberty:sa:2005-11"
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sa="urn:liberty:sa:2005-11"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:disco="urn:liberty:disco:2005-11"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns="urn:liberty:sa:2005-11"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="09">

<!-- Filename: lib-arch-authn-svc.xsd -->
<!-- Author: Jeff Hodges -->
<!-- Last editor: $Author: dchampagne $ -->
<!-- $Date: 2005/09/23 23:24:19 $ -->
<!-- $Revision: 1.10 $ -->

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

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

<x:import
namespace="urn:liberty:disco:2005-11"
schemaLocation="liberty-idwsf-disco-svc-v2.0.xsd"/>

<x:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>

<x:annotation>
  <xs:documentation>
  Liberty ID-WSF Authentication Service XSD
  </xs:documentation>
</xs:annotation>

<x:documentation>
The source code in this XSD file was excerpted verbatim from:
Liberty ID-WSF Authentication Service Specification
Version 2.0-08
22 Sep 2005
Copyright (c) 2005 Liberty Alliance participants,
see http://www.projectliberty.org/specs/idwsf_2_0_copyrights.php
</xs:documentation>

<x:element name="SASLRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Data" minOccurs="0">
        <xs:complexType>
          <xs:annotation>
            <xs:documentation>
            Liberty ID-WSF Authentication Service and Single Sign-On Service Specification
            </xs:documentation>
          </xs:annotation>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:simpleContent>
<xs:extension base="xs:base64Binary"/>
</xs:simpleContent>
</xs:complexType>
</xs:element>

<xselement ref="saml:RequestedAuthnContext" minOccurs="0"/>

</xs:sequence>
</xs:complexType>
</xs:element>

<xselement name="SASLResponse">
<xs:complexType>
<xs:sequence>
<xselement ref="Status"/>
<xselement ref="PasswordTransforms" minOccurs="0"/>
<xselement name="Data" minOccurs="0">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:base64Binary"/>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<!-- ID-WSF EPRs -->
<xselement ref="wsa:EndpointReference" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>
</xs:complexType>
</xs:element>

<!-- ID-WSF EPRs -->
<xselement ref="wsa:EndpointReference" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>
</xs:complexType>
</xs:element>

<!-- Password Transformations -->

<!-- Password Transformations -->
<xs:element name="PasswordTransforms">
  <xs:annotation>
    <xs:documentation>
      Contains ordered list of sequential password transformations
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Transform" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="name" type="xs:string" use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:element>
            <xs:attribute name="name" type="xs:anyURI" use="required"/>
            <xs:attribute name="id" type="xs:ID" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:complexType>
</xs:element>
</xs:schema>
D. liberty-idwsf-utility-v2.0.xsd Schema Listing

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="2.0-01">
<xs:include schemaLocation="liberty-utility-v2.0.xsd"/>
<xs:annotation>
<xs:documentation>
Liberty Alliance Project utility schema. A collection of common IDentity Web Services Framework (ID-WSF) elements and types. This schema is intended for use in ID-WSF schemas. This file intended for inclusion, rather than importation, into other schemas. This version: 2004-12
Copyright (c) 2005 Liberty Alliance participants, see http://www.projectliberty.org/specs/idwsf_2_0_r2_copyrights.php
</xs:documentation>
</xs:annotation>
</xs:schema>
E. liberty-utility-v2.0.xsd Schema Listing

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="2.0-01">
    <xs:annotation>
        <xs:documentation>
            Liberty Alliance Project utility schema. A collection of common elements and types for use with independent Liberty XML Schema documents.
            This file intended for inclusion, rather than importation, into other schemas.
            This version: 2004-12
            Copyright (c) 2004 Liberty Alliance participants, see http://www.projectliberty.org/specs/idff_copyrights.html
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType name="IDType">
        <xs:annotation>
            <xs:documentation>
                This type should be used to provide IDs to components that have IDs that may not be scoped within the local XML instance document.
            </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="IDReferenceType">
        <xs:annotation>
            <xs:documentation>
                This type can be used when referring to elements that are identified using an IDType.
            </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:complexType name="StatusType">
        <xs:annotation>
            <xs:documentation>
                A type that may be used for status codes.
            </xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="code" type="xs:string" use="required"/>
        <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
        <xs:attribute name="comment" type="xs:string" use="optional"/>
    </xs:complexType>
    <xs:element name="Status" type="StatusType">
        <xs:annotation>
            <xs:documentation>
                A standard Status type
            </xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="EmptyType">
        <xs:annotation>
            <xs:documentation> This type may be used to create an empty element </xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            </xs:complexContent>
        </xs:complexType>
    </xs:complexType>
</xs:schema>
<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>