Liberty IDP Service Specification

Version: 1.0

Editors:
Conor P. Cahill, Intel Corporation

Contributors:
Shelagh Callahan, Intel Corporation
Jane Dashevsky, Intel Corporation
Tapio Kaukonen, Nokia
Sampo Kellomäki, Symlabs, Inc.
Hubert Le Van Gong, Sun
Andrew Lindsay-Stewart, Vodafone
Paul Madsen, NTT
Paul Miller, Gemplus
Hiroyoshi Takiguchi, NTT
Pierre Vannel, Gemplus
Sean Walker, Axalto

Abstract:

This specification describes the ID-WSF IDP Service.

Filename: liberty-idwsf-idp-v1.0.pdf

Notice

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

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

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

Contents

1. Introduction

IdP Service Introduction

1.1. Notation and Conventions

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

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.

1.1.1. XML Namespaces

The following XML namespaces are referred to in this document:

- The prefix *idp:* represents the IdP Service namespace. This namespace is the default for instance fragments, type names, and element names in this document. In schema listings, and in examples of IdP Service messages and fragments thereof, this is the default namespace *when* no prefix is shown:

  urn:liberty:idp:2007-09

- The prefix *saml2:* stands for the SAMLv2 assertion namespace [SAMLCore2]:

  urn:oasis:names:tc:SAML:2.0:assertion

- The prefix *samlp2:* stands for the SAMLv2 protocol namespace [SAMLCore2]:

  urn:oasis:names:tc:SAML:2.0:protocol

- The prefix *xs:* stands for the W3C XML schema namespace [Schema1-2]:

  http://www.w3.org/2001/XMLSchema

- The prefix *xsi:* stands for the W3C XML schema instance namespace:

  http://www.w3.org/2001/XMLSchema-instance

2. Data Definitions

2.1. Common Provider ID

The URI *urn:liberty:idp:2007-09:ProviderID:Common* is a reserved URI which represents a globally shared identifier which may be used by clients in place of a unique *ProviderID* when the execution context of a request requires the privacy protections offered by the common identifier.

The Liberty ID-WSF Advanced Client Technologies Overview ([LibertyACT]) contains a complete discussion about the privacy and maintenance concerns surrounding the selection of a *ProviderID* for clients issuing authentication assertions.

2.2. Minting Assertion (MING)

The **minting assertion** (MING) is a SAML assertion issued by an IdP granting a TM permission to mint assertions on behalf of the IdP. This assertion will be built from standard SAML Assertion elements including:

- The `<saml2:Subject>` MUST be the TM, with the TM's ProviderID in the `<saml2:NameID>` sub-element and with one of the TM's public keys in the `<saml2:SubjeftConfirmation>` sub-element.

  An example `<Subject>`:

```
<saml2:Subject>
  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
    urn:liberty:idp:2007-09:ProviderID:Common
  </saml2:NameID>
  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
    <saml2:SubjectConfirmationData xsi:type="saml2:KeyInfoConfirmationDataType">
      <ds:KeyInfo> TM Public Key Info </ds:KeyInfo>
    </saml2:SubjectConfirmationData>
  </saml2:SubjectConfirmation>
</saml2:Subject>
```

  In this case, the TM is using the common provider ID (likely due to some privacy requirement).

- The SAML 2.0 standard `<saml2:Condition>`s apply to the MING itself (and indirectly impact the MED since the MED is not valid unless the MING is valid). In addition, there are new, MING-specific, conditions added to restrict some of the properties of the MED.

  The new condition types are:

  - `<idp:SubjectRestrictionType>` - this condition restricts the possible subjects that can be specified in any MED minted using this MING. The schema for the `<idp:SubjectRestrictionType>` is as follows:

```
<!-- SubjectRestriction - MING condition restricting Subjects in MED -->

<xs:complexType name="SubjectRestrictionType">
  <xs:complexContent>
    <xs:extension base="saml2:ConditionAbstractType">
      <xs:sequence>
        <xs:element ref="saml2:Subject"     maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

    Multiple `<saml2:Subject>`s MAY be specified allowing a single subject-restricted MING to be used for different MEDs for different subjects at different providers.

- `<idp:AuthnContextRestrictionType>` - this condition restricts the possible authentication contexts that can be specified in any MED minted using this MING. The schema for the `<idp:AuthnContextRestrictionType>` is as follows:

```
<!-- AuthnContextRestriction - Ming Condition - restricts authncontext in MED -->

<xs:complexType name="AuthnContextRestrictionType">
 <xs:complexContent>
  <xs:extension base="saml2:ConditionAbstractType">
   <xs:sequence>
    <xs:element ref="saml2:AuthnContext" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

  Multiple `<saml2:AuthnContext>`s MAY be specified allowing a single AuthnContext-restricted MING to be used for different MEDs with different AuthnContexts.

These new types are used by specifying a `<saml2:Conditions>` element with the `xsi:type` attribute set to one of the listed types above (and then the content of that `<saml2:Conditions>` element follows the definition of the extended type above.

- An `<saml2:AuthzDecisionStatement>` MUST be included with the following contents:
  - The `Resource` attribute MUST be set to "*urn:oasis:names:tc:SAML:2.0:assertion*".
  - The `Decision` attribute MUST be set to "*Permit*".
  - The `<Action>` element's `Namespace` attribute MUST be set to "*urn:liberty:idp:2007-09:Actions*" and the value of the element MUST be "*Mint*".

An example `<saml2:AuthzDecisionStatement>`:

```
<saml2:AuthzDecisionStatement
    Resource="urn:oasis:names:tc:SAML:2.0:assertion"
    Decision="Permit">
 <saml2:Action Namespace="urn:liberty:idp:2007-09:Actions">Mint</saml2:Action>
</saml2:AuthzDecisionStatement>
```

An example MING:

```
<saml2:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
            ID="sxJu9gvvLG9sAN9bKp8q0NKU"
            IssueInstant="2006-05-01T17:20:30.213Z"
            Version="2.0">

 <saml2:Issuer>http://idp.example.com</saml2:Issuer>
 <ds:Signature>IdP's signature data goes here</ds:Signature>

 <saml2:Subject>
  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
   urn:liberty:idp:2007-09:ProviderID:Common
  </saml2:NameID>
  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
   <saml2:SubjectConfirmationData>
    <ds:KeyInfo> PublicKey info for TM </ds:KeyInfo>
   </saml2:SubjectConfirmationData>
  </saml2:SubjectConfirmation>
 </saml2:Subject>

 <saml2:Conditions NotBefore="2006-05-01T17:20:30.213Z"
             NotOnOrAfter="2006-05-02T17:20:30.213Z">
```

```
    <saml2:AudienceRestriction>
      <saml2:Audience>Provider1</saml2:Audience>
      <saml2:Audience>Provider2</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>

  <saml2:AuthzDecisionStatement
     Resource="urn:oasis:names:tc:SAML:2.0:assertion"
     Decision="Permit">
    <saml2:Action Namespace="urn:liberty:idp:2007-09:Actions">Mint</saml2:Action>
  </saml2:AuthzDecisionStatement>
</saml2:Assertion>
```

Notes about this example:

- The TM is using a common ProviderID.
- The MING is restricted to either of 2 audiences.
- There is no Subject Restriction so the TM can use this MING for any valid subject it has for the specified relying parties.

An example MING with a subject and AuthnContext restrictions:

```
<saml2:Assertion  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
             ID="sxJu9gvvLG9sAN9bKp8q0NKU"
             IssueInstant="2006-05-01T17:20:30.213Z"
             Version="2.0">

  <saml2:Issuer>http://idp.example.com</saml2:Issuer>
  <ds:Signature>IdP's signature data goes here</ds:Signature>

  <saml2:Subject>
    <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
      urn:liberty:idp:2007-09:ProviderID:Common
    </saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
      <saml2:SubjectConfirmationData>
        <ds:KeyInfo> PublicKey info for TM </ds:KeyInfo>
      </saml2:SubjectConfirmationData>
    </saml2:SubjectConfirmation>
  </saml2:Subject>

  <saml2:Conditions NotBefore="2006-05-01T17:20:30.213Z"
               NotOnOrAfter="2006-05-02T17:20:30.213Z">
    <saml2:Condition xsi:type="idp:SubjectRestrictionType">
      <saml2:Subject>
        <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
          SPNameQualifier="Provider1">NameIDatProvider1</saml2:NameID>
      </saml2:Subject>
      <saml2:Subject>
        <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
          SPNameQualifier="Provider2">NameIDatProvider2</saml2:NameID>
      </saml2:Subject>
    </saml2:Condition>
    <saml2:Condition xsi:type="idp:AuthnContextRestrictionType">
      <saml2:AuthnContext>
        <saml2:AuthnContextClassRef>
          urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
        </saml2:AuthnContextClassRef>
      </saml2:AuthnContext>
      <saml2:AuthnContext>
        <saml2:AuthnContextClassRef>
          urn:oasis:names:tc:SAML:2.0:ac:classes:SmartcardPKI
```

```
        </saml2:AuthnContextClassRef>
      </saml2:AuthnContext>
    </saml2:Condition>
  </saml2:Conditions>

  <saml2:AuthzDecisionStatement
     Resource="urn:oasis:names:tc:SAML:2.0:assertion"
     Decision="Permit">
    <saml2:Action Namespace="urn:liberty:idp:2007-09:Actions">Mint</saml2:Action>
  </saml2:AuthzDecisionStatement>
</saml2:Assertion>
```

Notes about this example:

- The TM is using a CommonProviderID.
- The MING is restricted to either of 2 Subjects, one at Provider1 and one at Provider2.
- There is no `<saml2:AudienceRestriction>` necessary as the `<saml2:Subject>` elements included the `SPNameQualifier` attribute which has same result.
- The NameID elements were not encrypted. Normally, when NameIDs for multiple providers are included in an assertion that may be sent to multiple providers, an EncryptedID would be used. Doing so here would have made the example kind of big and ugly, so we didn't encrypt.
- The TM may generate MEDs that show the user authenticated with a password or with a Smartcard plus pin (the IdP is depending upon the TM to do the right thing when selecting which of those classes to use). Note that, in some cases, the TM, itself, can be within a Smartcard (or the equivalent) and this can be considered part of the authentication context of the user.

2.2.1. MING Creation Rules

When creating a MING, the following rules apply:

- All of the SAML 2.0 rules for the generation of an assertion MUST be observed.
- The `<saml2:Subject>` of the MING MUST contain the ProviderID of the TM and the `<saml2:SubjectConfirmation>` MUST be one that supports proof-of-possession via a digital signature (such as *urn:oasis:names:tc:SAML:2.0:cm:holder-of-key)*/
- If desired, include, at most, one `<idp:SubjectRestriction>` `<idp:SubjectRestriction>` and/or, at most, one `<idp:AuthnContextRestriction>` .
- If the assertion is to include a `<idp:SubjectRestriction>` condition and a `<saml2:AudienceRestriction>` condition, there SHOULD be at least one `<saml2:Subject>` in the `<idp:SubjectRestriction>` for every `<saml2:Audience>`.
- `<saml2:Subject>` elements within a `<idp:SubjectRestriction>` element MUST NOT include a `<saml2:SubjectConfirmation>` element.

2.2.2. MING Processing Rules

The processing rules for accepting a MING include:

- All of the SAML 2.0 rules for processing and accepting an assertion MUST be observed (e.g., if the current time is beyond the value in the `NotOnOrAfter` attribute on the `<saml2:Conditions>` element, the MING MUST be considered invalid).

- If the `<saml2:Conditions>` element contains a `<idp:SubjectRestriction>` element, the `<saml2:Subject>` of the MED (not the MING) MUST **strongly match** one of the `<saml2:Subject>`s within the `<idp:SubjectRestriction>`. If this is not the case, the MED MUST be considered invalid.

- If the `<saml2:Conditions>` element contains a `<idp:AuthnContextRestriction>` . element, the `<saml2:AuthnContext>` of the MED (not the MING) MUST match one of the `<saml2:AuthnContext>`s within the `<idp:AuthnContextRestriction>`. If this is not the case, the MED MUST be considered invalid.

- The `<saml2:Conditions>` on the MED (not MING) MUST be at least as restrictive as the `<saml2:Conditions>` on the MING (e.g., the MED MUST NOT have a `NotOnOrAfter` which is later than the `NotOnOrAfter` in the MING). If any of the conditions are less restrictive, the MED MUST be considered invalid.

2.3. Minted Assertion (MED)

The **minted assertion** (MED) is a SAML authentication assertion issued by a TM which includes a MING granting the TM permission to mint the MED. This assertion is built from standard SAML elements including:

- The `<saml2:Issuer>` MUST contain the TM's ProviderID. The possibilities for the TM ProviderID are discussed in the Client Identity section within the Liberty ID-WSF Advanced Client Technologies Overview ([LibertyACT]).

- The `<saml2:Subject>` SHOULD contain a `<saml2:NameID>` which identifies the principal (not the TM). The `NameQualifier` attribute SHOULD identify the IdP (as the identifier is issued in the namespace of the IdP, not the TM).

- The `<saml2:Conditions>` MAY contain any values permitted by the MING (including, for example, a validity period that is within the validity period of the MING).

- The MING MUST be included within the `<saml2:Advice>` element.

- All other SAML elements MAY be used, including Attribute Statements.

An example MED:

```
<saml2:Assertion  xmlns:saml="urn:oasis:names:tc:SAML: 2.0:assertion"
             ID="ID_1"
             IssueInstant="2006-05-02T11:43:12.723Z"
             Version="2.0">
  <saml2:Issuer>urn:liberty:idp:2007-0 9:ProviderID:Common</saml2: Issuer>
  <ds:Signature>
    TM's signature data goes here - MUST be signed using the
    the key identified by the keyinfo in the "MING"
  </ds:Signature>
  <saml2:Subject>
    <saml2:NameID
       NameQualifier="IdP's ProviderID"
       SPNameQualifier="SP's Provider ID"
       Format="urn:oasis:names:tc:SAML:2. 0:nameid-format:persistent">
      Principal's NameID at SP
    </saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML :2.0:cm:bearer">
      <saml2:SubjectConfirmationData InResponseTo="AuthnRequestID" />
    </saml2:SubjectConfirmation>
  </saml2:Subject>
  <saml2:Conditions NotBefore="2006-05-02T11:43:12.723Z"
```

```
                NotOnOrAfter="2006-05-02T11:48:12.723Z">
    <saml2:AudienceRestriction>
      <saml2:Audience>
        SP's Provider ID
      </saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:Advice>
    <saml2:Assertion ... >
      MING assertion data here
    </saml2:Assertion>
  </saml2:Advice>
  <saml2:AuthnStatement AuthnInstant="2006-05-02T11:43:12.723Z">
    Authentication information here
  </saml2:AuthnStatement>

</saml2:Assertion>
```

2.3.1. MED Creation Rules

When creating a MED, the following rules apply:

- All of the SAML 2.0 rules for the generation of an assertion MUST be observed.
- Restrict the `<saml2:Conditions>` and `<saml2:AuthnStatement>` element information to be within the limits contained within MING.
- Include the MING within the `<saml2:Advice>` element.
- Sign the assertion (the MED) using the private key associated with the public key identified in the `<saml2:SubjectConfirmation>` element of the MING.

2.3.2. MED Processing Rules

The processing rules for accepting a MED include:

- All of the SAML 2.0 rules for processing and accepting an assertion MUST be observed (e.g., if the current time is beyond the value in the `NotOnOrAfter` attribute on the `<saml2:Conditions>` element, the MED MUST be considered invalid).
- The relying party, seeing a `<saml2:Issuer>` it does not know, SHOULD locate the MING within the `<saml2:Advice>` and validate the MING independently. If the MING is not valid, the MED SHOULD NOT be considered valid.
- The MING must be present in the `<saml2:Advice>` and the MING itself MUST validate (also using all SAML2.0 rules for validation).
- If the MING `<saml2:Conditions>` element contains a `<idp:AuthnContextRestriction>` . element, the `<saml2:AuthnContext>` of the MED (not the MING) MUST match one of the `<saml2:AuthnContext>`s within the `<idp:AuthnContextRestriction>` in the MING. If this is not the case, the MED MUST be considered invalid.
- If any of the `<saml2:Conditions>` in the MED are LESS restrictive (e.g., a later `NotOnOrAfter`) than the `<saml2:Conditions>` in the MING, the MED MUST be considered invalid.

2.4. <ProviderInfo> element

The <ProviderInfo> element contains information describing a provider (name, ProviderID, etc.). This information is commonly used by clients to provide the user with a selection list of the providers with which the user may want to interact.

The <ProviderInfo> element contains the following attributes/elements:

- providerID [required]

  The ProviderID for the provider.

- name [optional]

  The name of the provider (typically used by the caller to allow the principal to select which providers that they anticipate using as the name is more understandable than the providerID).

The schema for the <ProviderInfo> element is shown below.

```
<!-- ProviderInfo - Information about providers returned by GetProviderInfo -->

<xs:element name="ProviderInfo" type="ProviderInfoType"/>

<xs:complexType name="ProviderInfoType">
  <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
  <xs:attribute name="name"     type="xs:string" use="optional"/>
</xs:complexType>

```

Figure 1. <ProviderInfo> — Schema Fragment

An example <ProviderInfo> element for the Acme Profile Service follows.

```
<idp:ProviderInfo providerID="http://services.acme.com/Profile"
          name="ACME Profile Service" />

```

Example 1. Example <ProviderInfo> Element

3. Identity Provider Service (IdP)

An IdP Service is a web service interface to an Identity Provider. This service provides interfaces which provide assertions and provider information to a client so that the client can facilitate SSO operations with relying parties.

An abstract WSDL definition for the IdP Service is included in this document, see Section 5: IDP Service WSDL . This WSDL document defines all of the "WSDL operations" for the IdP Service.

The complete schema for the IdP Service is included in this document, see Section 4: IDP Service Schema .

3.1. Service URIs

Table 1. IdP Service URIs

Use	URI
Service Type	*urn:liberty:idp:2007-09*
GetAssertion wsa:Action	*urn:liberty:idp:2007-09:GetAssertion*
GetProviderInfo wsa:Action	*urn:liberty:idp:2007-09:GetProviderInfo*

3.2. Status Codes

The following status code strings are defined:

- *OK*: message processing succeeded.
- *Failed*: general failure code.
- *Partial*: some portion of the request succeed and some failed. See the response element for more details.
- *NoResults*: the query had no matching results.
- *NotFound*: the specified item(s) were not found.

These strings are expected to appear in the "code" attribute of `<Status>` elements used in SOAP-bound IdP Service protocol messages [LibertySOAPBinding]. Specific uses for the status codes are defined in the processing rules for individual messages. The "ref" attribute on the `<Status>` element is not used in this specification, so it MUST NOT appear on Status elements in IdP Service protocol messages. The contents of the `comment` attribute are not defined by this specification, but it may be used for additional descriptive text intended for human consumption (for example, to carry information that will aid debugging).

3.3. Operation: *GetAssertion*

The *GetAssertion* operation is used by the TM to obtain Minting Assertions from the IdP.

3.3.1. `wsa:Action` values for `GetAssertion` Messages

`<GetAssertion>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:idp:2007-09:GetAssertion`."

`<GetAssertionResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:idp:2007-09:GetAssertionResponse`."

3.3.2. `GetAssertion` Message

The `<GetAssertion>` is called to obtain assertions for the TM. These assertions may be used for hoarding or minting depending upon the request parameters.

The `<GetAssertion>` contains the following attributes/elements:

- `purpose` [Required]

  The intended purpose for the requested assertions. The following two values are defined by this specification:

  - *urn:liberty:idp:2007-09:purpose:minting*

    The assertions issued in response to this request will be used for the purpose of creating (minting) authentication assertions by the caller.

    When this purpose is specified, the `<saml2p:AuthnRequest>` incorporates the following restrictions/interpretations:

    1. The `<saml2:Subject>` element MUST be specified and MUST ONLY include a `<saml2:SubjectConfirmation>` element.

       The identity of the subject MUST be taken from the *Target Identity* in the invocation context for the call.

    2. The subject confirmation method MUST be a method which requires proof of possession of a key (such as *urn:oasis:names:tc:SAML:2.0:cm:holder-of-key*).

    3. The `<saml2:SubjectConfirmationData>` element SHOULD carry the necessary information to identify the key that the TM plans to use to bind the minting assertion to any generated assertions associated with this provider.

    4. The `<saml2p:NameIDPolicy>` MAY be present.

5. The `<saml2:Conditions>` element MUST be present and MUST include at least one `<saml2:Audience>` element within the `<saml2:AudienceRestriction>` element. The `<saml2:Audience>` elements specify the provider ID(s) for the providers for which the TM desires a Minting Assertion.

   If multiple `<saml2:Audience>` elements are specified, the TM is requesting the necessary minting assertion(s) for the entire list of providers (which may be in multiple minting assertions).

   The `<saml2:Conditions>` element MAY also include other proposed conditions (such as `NotOnOrAfter`) that the TM would like to see reflected in the Minting Assertion(s) generated in response to this request. The IdP uses this data as input into its issuance algorithms and MAY choose conditions that are not fully in agreement with the proposed conditions.

6. The `<saml2p:RequestedAuthnContext>` MAY be present, but is interpreted differently. In this case, it specifies the AuthnContext(s) that the TM intends to use in MED(s). This gives the IdP guidance on what it may want to put into any `<idp:AuthnContextRestriction>` within the MINGs resulting from this request.

7. The `<saml2p:Scoping>` element SHOULD NOT be present.

8. The `ForceAuthn`, `IsPassive`, `AssertionConsumerServiceIndex`, `AssertionConsumerServiceURL`, `AttributeConsumingServiceIndex`, and `ProviderName`, attributes SHOULD NOT be present.

- *urn:liberty:idp:2007-09:purpose:SSO*

  The assertions issued in response to this request will be used for the purpose of single sign on operations (both browser-based and web-services-based) by the caller.

- `<saml2p:AuthnRequest>` [Any Number]

  The [SAMLCore2] authentication request containing the desired assertion properties.

  Multiple `<saml2p:AuthnRequest>` elements MAY be specified when necessary to describe the set of expected responses. Each such element is treated as an independent request with an associated `<idp:GetAssertionResponseItem>` element in the response.

- `Arbitrary Attributes`

  Arbitrary attributes may be added as necessary for a particular extension and/or inclusion of this element into a messaging protocol (such as an `xs:id` attribute).

The schema for the `<GetAssertion>` is shown below.

```
<!-- GetAssertion - get hoarding or minting assertions -->

<xs:element name="GetAssertion" type="GetAssertionType"/>

<xs:complexType name="GetAssertionType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="saml2p:AuthnRequest" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="purpose" type="xs:anyURI" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 2. `<GetAssertion>` — Schema Fragment

An example message body containing a <GetAssertion> message follows. This request gets Minting Assertions for 4 different providers using 3 different public keys. Note that for provider 1 and 2, the request allows a new federation handle to be created for the user.

```
<idp:GetAssertion purpose="urn:liberty:idp:2007-09:purpose:minting">

  <saml2p:AuthnRequest ID="ID_2343823023823" Version="2.0"
        IssueInstant="2006-06-23T15:38:46Z">
    <saml2:Subject>
      <saml2:SubjectConfirmation Method="...:holder-of-key">
        <saml2:SubjectConfirmationData>
          <ds:KeyInfo>Key info for Minting Assn (TM Public Key)</ds:KeyInfo>
        </saml2:SubjectConfirmationData>
      </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2p:NameIDPolicy Format="...:persistent" AllowCreate="true"/>
    <saml2:Conditions NotOnOrAfter="2006-07-23T15:38:46Z">
      <saml2:AudienceRestriction>
        <saml2:Audience>Provider 1</saml2:Audience>
        <saml2:Audience>Provider 2</saml2:Audience>
      </saml2:AudienceRestriction>
    </saml2:Conditions>
  </saml2p:AuthnRequest>

  <saml2p:AuthnRequest ID="ID_2343823023824" Version="2.0"
        IssueInstant="2006-06-23T15:38:46Z ">
    <saml2:Subject>
      <saml2:SubjectConfirmation Method="...:holder-of-key">
        <saml2:SubjectConfirmationData>
          <ds:KeyInfo>A second public key to be used by the TM</ds:KeyInfo>
        </saml2:SubjectConfirmationData>
      </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2p:NameIDPolicy Format="...:persistent"/>
    <saml2:Conditions NotOnOrAfter="2006-07-23T15:38:46Z">
      <saml2:AudienceRestriction>
        <saml2:Audience>Provider 3</saml2:Audience>
      </saml2:AudienceRestriction>
    </saml2:Conditions>
  </saml2p:AuthnRequest>

  <saml2p:AuthnRequest ID="ID_2343823023825" Version="2.0"
        IssueInstant="2006-06-23T15:38:46Z">
    <saml2:Subject>
      <saml2:SubjectConfirmation Method="...:holder-of-key">
        <saml2:SubjectConfirmationData>
          <ds:KeyInfo>A third public key to be used by the TM</ds:KeyInfo>
        </saml2:SubjectConfirmationData>
      </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2p:NameIDPolicy Format="...:persistent"/>
    <saml2:Conditions NotOnOrAfter="2006-07-23T15:38:46Z">
      <saml2:AudienceRestriction>
        <saml2:Audience>Provider 4</saml2:Audience>
      </saml2:AudienceRestriction>
    </saml2:Conditions>
  </saml2p:AuthnRequest>

</idp:GetAssertion>
```

Example 2. Example <GetAssertion> Message

3.3.3. GetAssertionResponse Message

This response to the <GetAssertion> request contains the following elements and attributes.

- <lu:Status> [required] - the completion status for the request. See the processing rules for more information.
- <idp:GetAssertionResponseItem> [optional] the result(s) of the <saml2p:AuthnRequest>(s). If the GetAssertion did not fail, there MUST be one <idp:GetAssertionResponseItem> in the response for each successful <saml2p:AuthnRequest> in the <idp:GetAssertion>. If partial results are being returned, the <idp:GetAssertionResponseItem> for failed <saml2p:AuthnRequest>s MUST NOT be included in the response.

  The <idp:GetAssertionResponseItem> element contains the following elements/attributes:

  - <idp:AssertionItem> [required] (one or more) - a container used to bind together assertions with optional MEDInfo (used when the assertions are MINGs). Multiple <idp:AssertionItem> elements MAY be included if the <saml2p:AuthnRequest> results in multiple assertions which have different MEDInfo parameters.

    This element consists of the following elements:

    - <idp:MEDInfo> [optional] Information deemed necessary for the generation of MEDs based on the MINGs in this <idp:GetAssertionResponseItem> at the TM. This is a container element that MUST contain the following elements:

      - <saml:NameID> [required] the principal's NameID at the provider. In the case where multiple providers are addressed by the same MING, multiple <saml:NameID>s will be present with the associated ProviderID in the SPNameQualifier.

        The TM MUST place the appropriate <saml:NameID> into the <saml:Subject> of any MEDs generated using the associated MING(s) (MINGs within the same <idp:AssertionItem> element).

        Note that though the schema does not require this element to be present (for validation reasons), **at least one** occurrence of this element MUST be present in the <idp:MEDInfo> element.

      - Other **[optional]** zero or more assertion statements (such as an <saml:AttributeStatement>). If present, the TM MUST place any such statements into any MED generated using this MING.

        This mechanism MUST NOT be used to insert statements whose generation is the responsibility of the TM such as the <saml:Subject>, <saml:Conditions>, or <saml:AuthnStatement>.

    - <saml2:Assertion> **[required]** - the assertion(s) resulting from the referenced <saml2p:AuthnRequest>.

    - created **[Optional]** - a boolean attribute set to *true* if the NameID in the enclosed assertion(s) and/or <idp:MEDInfo> element was newly created for this response

      If this attribute is not present, the assumed value is false.

      If the value is *true* the invoker of this <GetAssertion> MUST report whether or no this data was used in a transaction as soon as is reasonably possible following use of the data or expiration of the data, whichever comes first.

    - id **[Optional]** - an identifier for this assertion item. This attribute MUST be specified if the created attribute is *true*. The value of this attribute MUST be included in the ref attribute on any subsequent <CreatedStatus> request(s) reporting on the usage status of the newly created NameID.

  - ref: **[Required]** - the value from the ID attribute on the <saml2p:AuthnRequest> element in the request whose results are included in this <idp:AssertionItem> element.

- `Arbitrary Attributes`

  Arbitrary attributes may be added as necessary for a particular extension and/or inclusion of this element into a messaging protocol (such as an `xs:id` attribute).

```
<!-- GetAssertionResponse - the response for the GetAssertion request -->

<xs:element name="GetAssertionResponse" type="GetAssertionResponseType"/>

<xs:complexType name="GetAssertionResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="GetAssertionResponseItem"  minOccurs="0"
                                        maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="GetAssertionResponseItem" type="GetAssertionResponseItemType"/>
<xs:complexType name="GetAssertionResponseItemType">
  <xs:sequence>
    <xs:element ref="AssertionItem"    minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref"    type="xs:string"  use="required"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="AssertionItem" type="AssertionItemType"/>
<xs:complexType name="AssertionItemType">
  <xs:sequence>
    <xs:element ref="MEDInfo"         minOccurs="0"/>
    <xs:element ref="saml2:Assertion"   maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="created" type="xs:boolean" use="optional"/>
  <xs:attribute name="id"    type="xs:string"  use="optional"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="MEDInfo" type="MEDInfoType"/>
<xs:complexType name="MEDInfoType">
  <xs:sequence>
    <xs:any namespace="##other"       minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Figure 3. `<GetAssertionResponse>` — Schema Fragment

```
<idp:GetAssertionResponse>
  <lu:Status code="OK" />
  <idp:GetAssertionResponseItem ref="2343823023823">
    <idp:AssertionItem>
      <idp:MEDInfo>
        <saml2:NameID>..Provider 1 NameID...</saml2:NameID>
        <saml2:AttributeStatement>
          <saml2:Attribute Name="http://idp.com/MySpecialAttribute"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
            <saml2:AttributeValue>attribute-value</saml2:AttributeValue>
          </saml2:Attribute>
        </saml2:AttributeStatement>
      </idp:MEDInfo>
      <saml2:Assertion>..Provider 1 Minting Assertion.. </saml2:Assertion>
    </idp:AssertionItem>
    <idp:AssertionItem created="true">
      <idp:MEDInfo>
        <saml2:NameID>..Provider 2 NameID...</saml2:NameID>
      </idp:MEDInfo>
      <saml2:Assertion>..Provider 2 Minting Assertion.. </saml2:Assertion>
    </idp:AssertionItem>
  </idp:GetAssertionResponseItem>

  <idp:GetAssertionResponseItem ref="2343823023824">
    <idp:AssertionItem>
      <idp:MEDInfo>
        <saml2:NameID>..Provider 3 NameID...</saml2:NameID>
      </idp:MEDInfo>
      <saml2:Assertion>..Provider 3 Minting Assertion.. </saml2:Assertion>
    </idp:AssertionItem>
  </idp:GetAssertionResponseItem>

  <idp:GetAssertionResponseItem ref="2343823023825">
    <idp:AssertionItem>
      <idp:MEDInfo>
        <saml2:NameID>..Provider 4 NameID...</saml2:NameID>
      </idp:MEDInfo>
      <saml2:Assertion>..Provider 4 Minting Assertion.. </saml2:Assertion>
    </idp:AssertionItem>
  </idp:GetAssertionResponseItem>
</idp:GetAssertionResponse>
```

Example 3. `<GetAssertionResponse>` Message

3.3.4. GetAssertion Processing Rules

- Each `<saml2p:AuthnRequest>` MUST be processed independently of the others, succeeding or failing on its own. The results of each `<saml2p:AuthnRequest>` is returned in a `<idp:GetAssertionResponseItem>` element within the `<GetAssertionResponse>` message.

- The IdP MUST include exactly one `<idp:GetAssertionResponseItem>` in the `<GetAssertionResponse>` for each successful `<saml2p:AuthnRequest>` element in the request.

- If a particular `<saml2p:AuthnRequest>` is not successful, the IdP MUST NOT include a `<GetAssertionResponseItem>` referencing that `<saml2p:AuthnRequest>`

- The IdP may refuse to create new federations for the principal through this interface. In such cases the `<saml2p:AuthnRequest>` that would need a new federation to successfully complete MUST be treated as a failure. In such a case, if the IdP is providing second level error codes, the error code MUST be *NoCreated*.

- If the IdP is willing to create new federations for the principal but the `AllowCreate` attribute on the `<saml2p:NameIDPolicy>` is not set to *true*, the `<saml2p:AuthnRequest>` MUST be treated as a failure. In such a case, if the IdP is providing second level error codes, the error code MUST be *NotFederated*.

- If the `created` attribute is set to *true* on the `<idp:AssertionItem>`, the recipient MUST NOT use the data contained within as part an identity transaction with the relying party unless the relying party has indicated that it is willing to accept a newly federated identity (such as by setting the `AllowCreate` attribute on its `<saml2p:AuthnRequest>` to the recipient.

- If the IdP created a new persistent NameID to represent a new federation for the principal at the relying party:

  - The IdP MUST NOT treat this as a successfully completed federation until the IdP receives confirmation from the invoker that the new NameID has been used in a transaction.

    This is driven by the fact that, as far as the IdP knows, the Relying party did not request that the federation be generated (the invoker of this messages is not the relying party).

  - Should the IdP receive a federation request through another channel (such as a SAML2 AuthnRequest via browser SSO from the relying party) while the assertions resulting from this request are still valid, the IdP MUST use the same NameID.

    This prevents the IdP from issuing two different persistent NameIDs for the same principal at the same relying party.

- If the IdP created a new NameID to represent a new federation for the principal at the relying party, the IdP MUST not treat this as a successfully completed federation until the IdP receives confirmation from the invoker that the new NameID has been used in a transaction.

- The IdP MUST ONLY include `<idp:MedInfo>` elements in `<idp:AssertionItem>` elements that contain **minting assertions**.

- If the invoker's request for permission to mint assertions is not granted, the IdP must treat that `<saml2p:AuthnRequest>` as a failure and NOT return a `<GetAssertionResponseItem>` for that `<saml2p:AuthnRequest>` – in other words, the IDP MUST NOT generate a Minting assertion with an authorization decision of "Deny."

- If request processing succeeded, the top-level status code MUST be

  - *OK* - if **all**. `<saml2p:AuthnRequest>`s succeeded.

  - *Partial* - if some of the requests succeeded and some failed. The individual request status will be disclosed in the `<saml2p:Status>` element in each `<saml2p:Response>`.

  - *Failed* - if **all**. `<saml2p:AuthnRequest>`s failed.

- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* as a second-level status code. The IdP Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.4. Operation: *GetProviderInfo*

The *GetProviderInfo* operation is used to obtain Provider Information for providers with which the IdP has relationships.

3.4.1. `wsa:Action` values for `GetProviderInfo` Messages

`<GetProviderInfo>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:idp:2007-09:GetProviderInfo`."

`<GetProviderInfoResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:idp:2007-09:GetProviderInfoResponse`."

3.4.2. `GetProviderInfo` Message

The `<GetProviderInfo>` is called to obtain provider information about the providers with which an IdP has a relationship (so that the TM can select the providers with which it will interact).

The `<GetProviderInfo>` contains the following attributes/elements:

- `all` [optional]

  A boolean flag indicating whether the TM wants to get back all of the providers or just the providers with which the current user has a relationship.

  If present and set to "true," the IdP returns data for all available providers. Otherwise the IdP only returns the providers for which the principal has an established federation.

  If not present, the request is interpreted as if `all` was set to "false."

  This attribute has no effect if specific providers are requested using the `<ProviderID>` element.

- `<ProviderID>` [optional]

  zero or more Provider IDs for providers for whom the provider info is requested. If this element appears, the results are limited to the providers for which the IdP has a relationship which are also in this list.

The schema for the `<GetProviderInfo>` is shown below.

```
<!-- GetProviderInfo - Get list of providers visible to user/IdP -->

<xs:element name="GetProviderInfo" type="GetProviderInfoType"/>

<xs:complexType name="GetProviderInfoType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="all" type="xs:boolean" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="ProviderID" type="xs:anyURI"/>
```

Figure 4. `<GetProviderInfo>` — Schema Fragment

An example message body containing a `<GetProviderInfo>` message follows. This is a request for information about all of the providers associated with the current principal.

```
<idp:GetProviderInfo all="true"/>
```

Example 4. Example `<GetProviderInfo>` Message

3.4.3. `GetProviderInfoResponse` Message

This response to the `<GetProviderInfo>` request contains the following elements and attributes.

- `<lu:Status>`: Contains status code; see processing rules.
- One or more `<shps:ProviderInfo>` elements if the call did not fail.

```
<!-- GetProviderInfoResponse - response for the GetProviderInfo request -->

<xs:element name="GetProviderInfoResponse" type="GetProviderInfoResponseType"/>

<xs:complexType name="GetProviderInfoResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="ProviderInfo"       minOccurs="0"
                                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 5. `<GetProviderInfoResponse>` — Schema Fragment

```
<idp:GetProviderInfoResponse >
  <lu:Status code="OK" />
  <idp:ProviderInfo providerID="http://services.acme.com/Profile"
            name="ACME Profile Service" />
  <idp:ProviderInfo providerID="http://ps.services.Rus.com/"
            name="Services-R-Us People Service" />
</idp:GetProviderInfoResponse>
```

Example 5. `<GetProviderInfoResponse>` Message

3.4.4. GetProviderInfo Processing Rules

- If `<shps:ProviderID>` elements are specified on the request, the IdP MUST return `<shps:ProviderInfo>` elements for each provider in the intersection of the IdPs list of acceptable providers and the provided list of requested providers (only providers on both lists are included in the response).
- If no `<shps:ProviderID>` elements are specified in the request, the IdP should examine the `all` attribute. If this attribute is true, the IDP SHOULD return information about all providers on its list of acceptable providers.

  On the other hand, if the `all` attribute is not present or is set to "false," the IDP should only return information about providers which the principal has established federations with.
- The IDP MAY use other local policies to control and/or restrict the visibility of providers to the caller (including allowing the user to control this).
- If request processing succeeded, the top-level status code MUST be

- *OK* - if the call succeeded (which may include cases where no `<shps:ProviderID>` elements are returned – such as a principal which has not yet established any federations).

- *Failed* - if the call failed (because `<shps:ProviderID>` elements were specified on the request and **none** of them are known to the IdP.

- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* as a second-level status code. The IdP Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

3.5. Operation: *CreatedStatus*

The *CreatedStatus* operation is used to report to the IdP that a NameID which represents a new federation has been used.

3.5.1. `wsa:Action` values for `CreatedStatus` Messages

`<CreatedStatus>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:idp:2007-09:CreatedStatus`."

`<CreatedStatusResponse>` messages MUST include a `<wsa:Action>` SOAP header with the value of "`urn:liberty:idp:2007-09:CreatedStatusResponse`."

3.5.2. `CreatedStatus` Message

The `<CreatedStatus>` request is called to report the use of data from a `<GetAssertionResponse>` which had the `created` attribute set to *true*.

The `<CreatedStatus>` contains the following one or more `<CreatedStatusItem>` elements, each of which contain the following attributes/elements:

- `ref` **[required]** - a reference to the `<AssertionItem>` whose status is being reported. The value of the `id` attribute on the `<AssertionItem>` is specified here.

- `used` **[required]** - the status of the item (whether or not it was used). If the assertion was used this attribute MUST be set to *true* and otherwise it MUST be set to *false*.

- `firstUsed` **[optional]** - the date/time when the data was first used.

The schema for the <CreatedStatus> is shown below.

```
<!-- CreatedStatus - report on use of a new federation -->

<xs:element name="CreatedStatus" type="CreatedStatusType"/>

<xs:complexType name="CreatedStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="CreatedStatusItem" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="CreatedStatusItem" type="CreatedStatusItemType"/>

<xs:complexType name="CreatedStatusItemType">
  <xs:attribute name="ref"      type="xs:anyURI"  use="required"/>
  <xs:attribute name="used"     type="xs:boolean"  use="required"/>
  <xs:attribute name="firstUsed"  type="xs:dateTime" use="optional"/>
</xs:complexType>
```

Figure 6. <CreatedStatus> — Schema Fragment

An example message body containing a <CreatedStatus> message follows. This is a reporting on the use of a <saml2:NameID> that had been returned as part of a MING response.

```
<idp:CreatedStatus>
  <idp:CreatedStatusItem used="true"
    ref="uuid:9230230-238023-2309238-83494" firstUsed="2007-01-11T22:17:37Z"/>
</idp:CreatedStatus>
```

Example 6. Example <CreatedStatus> Message

3.5.3. CreatedStatusResponse Message

This response to the <CreatedStatus> request contains the following elements and attributes.

- <lu:Status>: Contains status code; see processing rules.

```
<!-- CreatedStatusResponse - response to the CreatedStatus request -->

<xs:element name="CreatedStatusResponse" type="CreatedStatusResponseType" />

<xs:complexType name="CreatedStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>
```

Figure 7. `<CreatedStatusResponse>` — Schema Fragment

```
<idp:CreatedStatusResponse>
  <lu:Status code="OK"/>
</idp:CreatedStatusResponse>
```

Example 7. `<CreatedStatusResponse>` Message

3.5.4. CreatedStatus Processing Rules

- Upon receipt of this call, the IDP should consider the federation process to be complete and should maintain the persistent identifier as appropriate.
- If the specified `<saml2:NameID>` or `<saml2:Assertion>` were not issued by the IdP to this invoker, the request MUST be treated as a failure. In such cases, if the IdP is reporting second level status codes, the error code MUST be *NotIssued*.
- If request processing succeeded, the top-level status code MUST be
  - *OK* - if the call succeeded.
  - *Partial* - if some of the requests succeeded and some failed. The individual request status will be disclosed in the `<saml2p:Status>` element in each `<saml2p:Response>`.
  - *Failed* - if all of the `<idp:CreatedStatusItem>` elements failed.
- If the top-level status code is *Failed*, the response MAY also contain *Forbidden* as a second-level status code. The IdP Service instance may not wish to reveal the reason for failure, in which case no second-level status code will appear.

4. IDP Service Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:idp:2007-09"
   xmlns:lu="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
   xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
   xmlns="urn:liberty:idp:2007-09"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>

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

  <xs:import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
    schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
    schemaLocation="http://docs.oasis-open.org/security/saml/v2.0/saml-schema-protocol-2.0.xsd"/>

  <!-- Data Definitions { -->



  <!-- ProviderInfo - Information about providers returned by GetProviderInfo -->

  <xs:element name="ProviderInfo" type="ProviderInfoType"/>

  <xs:complexType name="ProviderInfoType">
    <xs:attribute name="providerID" type="xs:anyURI" use="required"/>
    <xs:attribute name="name"     type="xs:string" use="optional"/>
  </xs:complexType>


  <!-- AuthnContextRestriction - Ming Condition - restricts authncontext in MED -->

  <xs:complexType name="AuthnContextRestrictionType">
    <xs:complexContent>
      <xs:extension base="saml2:ConditionAbstractType">
        <xs:sequence>
          <xs:element ref="saml2:AuthnContext" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- SubjectRestriction - MING condition restricting Subjects in MED -->

  <xs:complexType name="SubjectRestrictionType">
    <xs:complexContent>
      <xs:extension base="saml2:ConditionAbstractType">
        <xs:sequence>
          <xs:element ref="saml2:Subject"     maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>



  <!-- End of  Data Definitions } -->

  <!-- Interface Definitions { -->


```

```
<!-- GetAssertion - get hoarding or minting assertions -->

<xs:element name="GetAssertion" type="GetAssertionType"/>

<xs:complexType name="GetAssertionType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="saml2p:AuthnRequest" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="purpose" type="xs:anyURI" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- GetAssertionResponse - the response for the GetAssertion request -->

<xs:element name="GetAssertionResponse" type="GetAssertionResponseType"/>

<xs:complexType name="GetAssertionResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="GetAssertionResponseItem"  minOccurs="0"
                                      maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="GetAssertionResponseItem" type="GetAssertionResponseItemType"/>
<xs:complexType name="GetAssertionResponseItemType">
  <xs:sequence>
    <xs:element ref="AssertionItem"    minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref"    type="xs:string"  use="required"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="AssertionItem" type="AssertionItemType"/>
<xs:complexType name="AssertionItemType">
  <xs:sequence>
    <xs:element ref="MEDInfo"         minOccurs="0"/>
    <xs:element ref="saml2:Assertion"   maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="created" type="xs:boolean" use="optional"/>
  <xs:attribute name="id"    type="xs:string"  use="optional"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:element name="MEDInfo" type="MEDInfoType"/>
<xs:complexType name="MEDInfoType">
  <xs:sequence>
    <xs:any namespace="##other"       minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>



<!-- GetProviderInfoResponse - response for the GetProviderInfo request -->

<xs:element name="GetProviderInfoResponse" type="GetProviderInfoResponseType"/>

<xs:complexType name="GetProviderInfoResponseType">
  <xs:complexContent>
```

```
    <xs:extension base="ResponseAbstractType">
      <xs:sequence>
        <xs:element ref="ProviderInfo"       minOccurs="0"
                                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<!-- GetProviderInfo - Get list of providers visible to user/IdP -->

<xs:element name="GetProviderInfo" type="GetProviderInfoType"/>

<xs:complexType name="GetProviderInfoType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="ProviderID" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="all" type="xs:boolean" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="ProviderID" type="xs:anyURI"/>


<!-- CreatedStatus - report on use of a new federation -->

<xs:element name="CreatedStatus" type="CreatedStatusType"/>

<xs:complexType name="CreatedStatusType">
  <xs:complexContent>
    <xs:extension base="RequestAbstractType">
      <xs:sequence>
        <xs:element ref="CreatedStatusItem" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="CreatedStatusItem" type="CreatedStatusItemType"/>

<xs:complexType name="CreatedStatusItemType">
  <xs:attribute name="ref"      type="xs:anyURI"   use="required"/>
  <xs:attribute name="used"     type="xs:boolean"  use="required"/>
  <xs:attribute name="firstUsed"  type="xs:dateTime" use="optional"/>
</xs:complexType>


<!-- CreatedStatusResponse - response to the CreatedStatus request -->

<xs:element name="CreatedStatusResponse" type="CreatedStatusResponseType"/>

<xs:complexType name="CreatedStatusResponseType">
  <xs:complexContent>
    <xs:extension base="ResponseAbstractType" />
  </xs:complexContent>
</xs:complexType>


<!-- RequestAbstractType - common request message structure -->

<xs:complexType name="RequestAbstractType" abstract="true">
    <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

```
  <!-- ResponseAbstractType - common message response structure -->

  <xs:complexType name="ResponseAbstractType" abstract="true">
    <xs:sequence>
       <xs:element ref="lu:Status"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>



  <!-- End of Interface Definitions } -->

</xs:schema>
```

5. IDP Service WSDL

```
<?xml version="1.0"?>
<definitions name="idp-svc"
 targetNamespace="urn:liberty:idp:2007-09"
 xmlns:tns="urn:liberty:idp:2007-09"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
 xmlns:idp="urn:liberty:idp:2007-09"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
                http://schemas.xmlsoap.org/wsdl/
    http://www.w3.org/2006/02/addressing/wsdl
    http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

 <types>
  <xsd:schema>
   <xsd:import namespace="urn:liberty:idp:2007-09"
            schemaLocation="liberty-idwsf-idp-v1.0.xsd"/>
  </xsd:schema>
 </types>

 <message name="GetAssertion">
  <part name="body" element="idp:GetAssertion"/>
 </message>
 <message name="GetAssertionResponse">
  <part name="body" element="idp:GetAssertionResponse"/>
 </message>

 <message name="GetProviderInfo">
  <part name="body" element="idp:GetProviderInfo"/>
 </message>
 <message name="GetProviderInfoResponse">
  <part name="body" element="idp:GetProviderInfoResponse"/>
 </message>

 <message name="CreatedStatus">
  <part name="body" element="idp:CreatedStatus"/>
 </message>
 <message name="CreatedStatusResponse">
  <part name="body" element="idp:CreatedStatusResponse"/>
 </message>

 <portType name="IDPPort">

  <operation name="GetAssertion">
   <input message="tns:GetAssertion"
     wsaw:Action="urn:liberty:idp:2007-09:GetAssertion" />
   <output message="tns:GetAssertionResponse"
    wsaw:Action="urn:liberty:idp:2007-09:GetAssertionResponse" />
  </operation>

  <operation name="GetProviderInfo">
   <input message="tns:GetProviderInfo"
     wsaw:Action="urn:liberty:idp:2007-09:GetProviderInfo" />
   <output message="tns:GetProviderInfoResponse"
    wsaw:Action="urn:liberty:idp:2007-09:GetProviderInfoResponse" />
  </operation>

  <operation name="CreatedStatus">
   <input message="tns:CreatedStatus"
     wsaw:Action="urn:liberty:idp:2007-09:CreatedStatus" />
   <output message="tns:CreatedStatusResponse"
    wsaw:Action="urn:liberty:idp:2007-09:CreatedStatusResponse" />
  </operation>
```

```
  </portType>

  <!--
  An example of a binding and service that can be used with this
  abstract service description is idpided below.
  -->

  <binding name="IDPBinding" type="tns:IDPPort">

    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="GetAssertion">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="GetProviderInfo">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

    <operation name="CreatedStatus">
      <input> <soap:body use="literal"/> </input>
      <output> <soap:body use="literal"/> </output>
    </operation>

  </binding>

  <service name="IDPService">

    <port name="IDPPort" binding="tns:IDPBinding">

      <!-- Modify with the REAL SOAP endpoint -->

      <soap:address location="http://example.com/idp"/>

    </port>

  </service>

</definitions>
```

References

Normative

[LibertyACT] Cahill, Conor P., eds. "Liberty Advanced Client Technologies," Version 1.0, Liberty Alliance Project (14 December 2007). *http://www.projectliberty.org/specs*

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

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

[LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertyReg] Kemp, John, eds. "Liberty Enumeration Registry Governance," Version 1.1, Liberty Alliance Project (14 December, 2004). *http://www.projectliberty.org/specs*

[LibertySOAPAuthn] Hodges, Jeff, Aarts, Robert, Madsen, Paul, Cantor, Scott, eds. " Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification ," v2.0-errata-1.0-01, Liberty Alliance Project (28 November, 2006). *http://www.projectliberty.org/specs*

[LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

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

[RFC2251] "Lightweight Directory Access Protocol (v3)," M. Wahl T. Howes S. Kille (December 1997). RFC 2251, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2251.txt*

[RFC2252] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," M. Wahl A. Coulbeck T. Howes S. Kille (December 1997). RFC 2252, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2252.txt*

[RFC2891] Howes, T., Wahl, M., eds. (August 2000). "LDAP Control Extension for Server Side Sorting of Search Results," RFC 2891, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2891.txt*

[SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf*

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

[SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler, Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf*

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

[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/*

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

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

[XPATH] Clark , J., DeRose , S., eds. (16 November 1999). "XML Path Language (XPath) Version 1.0 ," Recommendation, W3C *http://www.w3.org/TR/xpath [August 2003].*

Informative

[LibertyIDWSFGuide] Weitzel, David, eds. "Liberty ID-WSF Implementation Guide," Version 2.0-02, Liberty Alliance Project (13 January, 2005). *http://www.projectliberty.org/specs*

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

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