Liberty ID-FF Protocols and Schema Specification

Version: 1.2-errata-v3.0

Editors:
Scott Cantor, Internet2, The Ohio State University
John Kemp, IEEE-ISTO

Contributors:
Robert Aarts, Nokia Corporation
John Beatty, Sun Microsystems, Inc.
Conor Cahill, AOL Time Warner, Inc.
Xavier Serret, Gemplus SA
Greg Whitehead, Trustgenix, Inc.

Abstract:

This specification defines a core set of protocols that collectively provide a solution for identity federation management, cross-domain authentication, and session management. This specification contains the core protocols and schema for Liberty identity federation. The reader is presumed to be generally familiar with the SAML specifications.

Filename: draft-liberty-idff-protocols-schema-1.2-errata-v3.0.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.** Implementors of this Specification are advised to review the Liberty Alliance Project's website (http://www.projectliberty.org/) for information concerning any Necessary Claims Disclosure Notices that have been received by the Liberty Alliance Management Board.

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

Contents

1. Introduction

This specification defines the abstract Liberty protocols for identity federation, single sign-on, name registration, federation termination, and single logout. Several concrete bindings and profiles of these protocols are defined in [LibertyBindProf].

1.1. Notation

This specification uses schema documents conforming to W3C XML Schema (see [Schema1]) and normative text to describe the syntax and semantics of XML-encoded SAML assertions and protocol messages. Note: Phrases and numbers in brackets [ ] refer to other documents; details of these references can be found in Section 5 (at the end of this document).

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT,", "RECOMMENDED," "MAY," and "OPTIONAL" in this specification 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.

```
Listings of schemas appear like this.
```

```
Listings of instance fragments appear like this.
```

The following namespaces are referred to in this document:

The prefix `lib:` stands for the Liberty ID-FF namespace (`urn:liberty:iff:2003-08`). This namespace is the default for instance fragments, type names, and element names in this document.

The prefix `ac:` stands for the Liberty authentication context namespace (`urn:liberty:ac:2003-08`)

The prefix `md:` stands for the Liberty metadata namespace (`urn:liberty:metadata:2003-08`)

The prefix `saml:` stands for the SAML 1.1 assertion namespace (`urn:oasis:names:tc:SAML:1.0:assertion`).

The prefix `samlp:` stands for the SAML 1.1 protocol namespace (`urn:oasis:names:tc:SAML:1.0:protocol`).

The prefix `ds:` stands for the W3C XML signature namespace (`http://www.w3.org/2000/09/xmldsig#`).

The prefix `xenc:` stands for the W3C XML encryption namespace (`http://www.w3.org/2001/04/xmlenc#`).

The prefix `xsd:` stands for the W3C XML schema namespace (`http://www.w3.org/2001/XMLSchema`). In schema listings, this is the default namespace and no prefix is shown.

The prefix `xsi:` stands for the W3C XML schema instance namespace (`http://www.w3.org/2001/XMLSchema-instance`).

This specification uses the following typographical conventions in text: `<Element>`, `<ns:ForeignElement>`, `Attribute`, **`Datatype`**, `OtherCode`.

For readability, when an XML Schema type is specified to be **`xsd:boolean`**, this document discusses the values as "true" and "false" rather than the "1" and "0" which are also legal **`xsd:boolean`** values.

Definitions for Liberty-specific terms can be found in [LibertyGlossary].

1.2. Overview

This specification defines a set of protocols that collectively provide a solution for identity federation management, cross-domain authentication, and session management.

The Liberty architecture contains three actors: Principal, identity provider, and service provider. A Principal is an entity (for example, an end user) that has an identity provided by an identity provider. A service provider provides services to the Principal.

Once the Principal is authenticated to the identity provider, the identity provider can provide an authentication assertion to the Principal, who can present the assertion to the service provider. The Principal is then also authenticated to the service provider if the service provider trusts the assertion. An identity federation is said to exist between an identity provider and a service provider when an identity provider issues assertions with a persistent name identifier regarding a particular Principal to the service provider. This specification defines a protocol where the identity of the Principal can be federated between the identity provider and the service provider. service providers can also request a non-persistent, one-time only, anonymous name identifier for the Principal.

This specification relies on the SAML specification in [SAMLCore11]. In SAML terminology, an identity provider acts as an Asserting Party and an Authentication Authority, while a service provider acts as a Relying Party.

2. Schema Declarations

This document specifies an XML schema for Liberty ID-FF. The schema header along with namespace, type, and element declarations are in 2.1 and 2.2.

2.1. Schema Header and Namespace Declarations

The following schema fragment defines the XML namespaces and other header information for the Liberty schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:iff:2003-08"
   xmlns="urn:liberty:iff:2003-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:ac="urn:liberty:ac:2003-08"
   xmlns:md="urn:liberty:metadata:2003-08"
   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   elementFormDefault="qualified" attributeFormDefault="unqualified">

   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
      schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
      schemaLocation="oasis-sstc-saml-schema-protocol-1.1.xsd"/>
   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
   <xs:import namespace="urn:liberty:ac:2003-
08" schemaLocation="liberty-authentication-context-v1.3.xsd"/>
   <xs:import namespace="urn:liberty:metadata:2003-0
8" schemaLocation="liberty-metadata-v1.1.xsd"/>

   <xs:include schemaLocation="liberty-idff-utility-v1.0.xsd"/>

   <xs:annotation>
      <xs:documentation> ### IMPORTANT NOTICE ###

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

Liberty Protocols and Schema Specification
Version 1.2-errata-v3.0
19 May 2005
Updated for WSF 1.1 release

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

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

2.1.1. Type and Element Declarations

Declarations for types and elements that are subsequently referred to in this document are as follows:

```
   <xs:element name="ProviderID" type="md:entityIDType"/>
   <xs:element name="AffiliationID" type="md:entityIDType"/>
```

3. Protocols

The Liberty protocol suite consists of the following protocols:

Single Sign-On and Federation: The protocol by which identities are federated and by which single sign-on occurs.

Name Registration: The protocol by which a provider can register an alternative opaque handle (or name identifier) for a Principal.

Federation Termination Notification: The protocol by which a provider can notify another provider than a particular identity federation has been terminated (also known as de-federation).

Single Logout: The protocol by which providers notify each other of logout events.

Name Identifier Mapping: The protocol by which service providers can obtain (often encrypted) name identifiers corresponding to an identity federation in which they do not participate.

3.1. General Requirements

The following sections define a set of general requirements applicable to all protocols.

3.1.1. XML Signature

The XML signature specification calls out a general XML syntax for signing data. All signed XML entities MUST adhere to the "XML Signature Profile" constraints defined in [SAMLCore11].

3.1.2. Protocol and Assertion Versioning

Version information appears in protocol messages and assertions defined in this specification. This specification defines version 1.2 for Liberty protocol messages and assertions. Version numbering of assertions is independent of the version numbering of the protocol messages. Any Liberty-defined elements in this specification containing `MajorVersion` and `MinorVersion` attributes MUST contain the values 1 and 2 respectively.

In other respects, this specification follows the version numbering requirements, processing rules, and error conditions specified in "SAML Versioning" in [SAMLCore11]. This means that any SAML-defined elements used in this specification containing `MajorVersion` and `MinorVersion` attributes MUST contain the values 1 and 1 respectively.

Most protocol messages and assertions used in the protocols defined in this specification are defined in the Liberty ID-FF namespace, and are therefore assigned the 1.2 version designation. A notable exception is when the SSO artifact profile is used, in which case pure SAML 1.1 `Request` and `Response`, elements are exchanged when dereferencing the artifact. These messages have a 1.1 version designation because they are SAML 1.1 protocol messages. In contrast, the `Assertion` element inside the message is a Liberty assertion and therefore carries the 1.2 designation.

In assigning version attributes, implementers should follow these rules:

- If the XML element containing the `MajorVersion` and `MinorVersion` attributes is in the Liberty ID-FF namespace (`urn:liberty:iff:2003-08`) or has an `xsi:type` attribute value in this namespace, then the `MajorVersion` MUST be 1 and the `MinorVersion` MUST be 2.

- If the element or its type is in a SAML 1.1 namespace (`urn:oasis:names:tc:SAML:1.0:assertion` or `urn:oasis:names:tc:SAML:1.0:protocol`), then the values MUST be 1 and 1 respectively.

3.1.3. Provider and Affiliation ID Uniqueness

All providers and affiliations have a URI-based identifier. A provider's URI-based identifier MUST be unique within the scope of all providers with which it communicates. It is RECOMMENDED that a provider use a URL with its own domain name for this identifier. Any URI-based identifier MUST NOT be more than 1024 characters in length.

All provider and affiliation identifiers MUST conform to the rules specified in [LibertyMetadata] regarding such identifiers.

Some profiles of the protocols contained in this specification may require a succinct 20-byte identifier. A provider MUST derive any such identifier by generating the SHA-1 hash of its URI-based identifier.

3.1.4. Name Identifier Construction

Principals are assigned name identifiers by identity providers and potentially by service providers. When generated by the identity provider, a name identifier MUST be constructed using pseudo-random values that have no discernible correspondence with the Principal's identifier (e.g., username) at the identity provider. The intent is to create a non-public pseudonym to prevent the discovery of the Principal's identity or activities. Service providers SHOULD follow the same construction rules. Unencrypted name identifier values MUST NOT exceed a length of 256 characters.

When generating one-time-use identifiers for Principals, in the case that a pseudorandom technique is employed, the probability of two randomly chosen identifiers being identical MUST be less than or equal to 2^{-128} and SHOULD be less than or equal to 2^{-160}. These levels correspond, respectively, to use of strong 128-bit and 160-bit hash functions, in conjunction with sufficient input entropy.

3.1.5. Signature Verification

Processing rules for the protocols defined in this document commonly specify digital signature verification. In these cases, it is not sufficient to only verify the signature of the signed object. The processing rules defined in [SAMLCore11] apply to all signed Liberty protocol messages. Verification of the `<ds:Signature>` element MUST be performed in accordance with the best practices for the certification path technology in use. For example, when using X.509 v3 public key certificates it is strongly RECOMMENDED that certification path validation be performed in accordance to the PKIX Profile as specified in [RFC3280].

3.1.6. Security

Because this specification defines only abstract protocols and does not define specific protocol profiles or the environment in which protocols will be deployed, most security requirements are deferred to individual profiles. See [LibertyBindProf] for security considerations for the Liberty-defined bindings and profiles. When a general security requirement can be stated for one of the abstract protocols described in this specification, the requirement is stated in line with the specific protocol.

3.1.7. Time Values

All Liberty time values have the type **`dateTime`**, which is built into the W3C XML Schema Datatypes specification [Schema2]. Liberty time values MUST be expressed in UTC form, indicated by a "Z" immediately following the time portion of the value.

Liberty requesters and responders SHOULD NOT rely on other applications supporting time resolution finer than seconds, as implementations MAY ignore fractional second components specified in timestamp values. Implementations MUST NOT generate time instants that specify leap seconds.

3.1.8. Time Synchronization

Providers SHOULD NOT assume that other providers have clocks that are synchronized closer than one minute.

The identity provider SHOULD NOT include a `NotBefore` attribute on the Conditions element of the assertion it generates which contains the time the assertion was generated.

The identity provider SHOULD NOT include a `NotOnOrAfter` attribute on the Conditions element of the assertion it generates which is less than one minute later than the time when the assertion was generated.

The service provider SHOULD NOT terminate the principal's session based solely on the `NotOnOrAfter` attribute of the Conditions element of the assertion used to authenticate the principal. If the assertion was valid when the principal was authenticated, the Principal SHOULD remain authenticated until one of the following occurs:

- `<LogoutRequest>` is received
- The user's session times out via normal means
- The `ReauthenticateOnOrAfter` time on the `<AuthenticationStatement>` used to authenticate the Principal, if any, is reached

3.1.9. Response Status Codes

All Liberty response messages use `<samlp:StatusCode>` elements to indicate the status of a corresponding request. Responders MUST comply with the rules governing `<samlp:StatusCode>` elements specified in [SAMLCore11] regarding the use of nested second-, or lower-level response codes to provide specific information relating to particular errors. A number of status codes are defined within the Liberty namespace for use with this specification.

3.1.10. Use of <Extension> in Protocols

Most of the protocol messages defined in this document contain a generic `<Extension>` element that permits the inclusion of arbitrary XML content representing agreements between providers that go beyond the bounds of the specification.

Implementers should understand that while extension content can be of a complex nature when fully XML-capable profiles are used, this is not the case for profiles that bind protocol messages to a URL query string. When using such profiles, the extension content MUST be deterministically expressible as a sequence of name/value pairs. This requires that the XML content MUST be confined to attributes and simple element content in the "null" namespace with non-overlapping local names. The total size of extension content SHOULD be minimized.

3.1.11. Interoperation with Previous Liberty Implementations

The protocols and schema definitions in this document are not compatible with previous versions of this specification. The following guidelines will assist implementers and deployers of the 1.2 specification in maximizing the opportunities for interoperability with software that implements older versions of the specification. The primary goal is to avoid sending messages to communication endpoints which those endpoints will not understand.

- Metadata SHOULD be used to the greatest extent possible to identity the capabilities of a provider, so that the proper messages can be sent. See [LibertyMetadata].
- Version 1.2 implementations SHOULD avoid sending 1.2 requests or notifications to providers that only implement earlier versions of the specification.

- When in doubt, 1.2 implementations SHOULD send 1.1 requests and notifications when the older specification meets the requirements of the transaction.

- Version 1.2 implementations MUST respond to older requests with responses matching the version of the request.

- These rules apply to all of the request/response protocols and asynchronous notifications defined in this specification.

3.1.11.1. Deprecation of ID-FF 1.1 Name Identifier Practices

Previous ID-FF specifications were not explicit about the use (or lack of use) of the SAML `Format` and `NameQualifier` XML attributes in Liberty `<NameIdentifier>` elements. Whereas this specification explicitly profiles the use of those attributes in a strict fashion, previous implementations have been found to make use of these attributes for their own purposes. Convention has been for implementations to reproduce whatever values they received from other implementations.

This specification deprecates the non-standard use of these attributes and implementations MUST follow the rules laid out in this specification in communicating all new identity federations created between version 1.2 providers. However, 1.2 providers SHOULD maintain interoperability with older providers by honoring and reproducing non-standard values for these attributes when communicating with older providers regarding existing identity federations. But, it is RECOMMENDED that 1.2 implementations provide tools by which these older identity federations can be "upgraded" to the behavior outlined in this specification by means of the Name Registration Protocol when both parties to a federation support the 1.2 specification (see Section 3.3).

Finally, when establishing new federated identifiers with 1.1 or earlier providers, 1.2 implementations MUST omit the `Format` and `NameQualifier` attributes in the 1.1 protocol messages they send. This is possible because all such identifiers have federated semantics and do not involve affiliations. Thus, the presence of the format values in this specification unambiguously indicates a 1.2 federation and the absence of those values indicates a pre-1.2 federation in all protocol messages.

3.1.12. Use of the consent attribute

In messages where a consent attribute is specified, this attribute should be used to indicate whether or not a user's consent has been obtained by the message sender.

The following values are defined for this attribute:

- *urn:liberty:consent:obtained* indicates that a user's consent has been obtained by the sender of the message. If the message sender uses this value, they SHOULD sign the message such that the signature covers this attribute.

- *urn:liberty:consent:obtained:prior* indicates that a user's consent has been obtained by the sender of the message at some point prior to this action. If the message sender uses this value, they SHOULD sign the message such that the signature covers this attribute.

- *urn:liberty:consent:obtained:current:implicit* indicates that a user's consent has been implicitly obtained by the sender of the message during this action as part of a broader indication of consent. If the message sender uses this value, they SHOULD sign the message such that the signature covers this attribute. Implicit consent is typically more proximal to the action in time and presentation than prior consent, such as part of a session of activities.

- *urn:liberty:consent:obtained:current:explicit* indicates that a user's consent has been explicitly obtained by the sender of the message during this action. If the message sender uses this value, they SHOULD sign the message such that the signature covers this attribute.

- *urn:liberty:consent:unavailable* indicates that the message sender did not obtain consent.

- *urn:liberty:consent:inapplicable* indicates that the message sender does not believe that they need to obtain or report consent in the sending of this message.

3.2. Single Sign-On and Federation Protocol

The Single Sign-On and Federation Protocol defines a request and response protocol by which single sign-on and identity federation occurs. The protocol is conducted between a service provider and one or more identity providers. The protocol works as follows (note that step 1 is optional):

1. A service provider issues an `<AuthnRequest>` to an identity provider, instructing the identity provider to provide an authentication assertion to the service provider. Optionally, the service provider MAY request that the identity be federated.

2. The identity provider responds with either an `<AuthnResponse>` containing authentication assertions to the service provider or an artifact that can be de-referenced into an authentication assertion. Additionally, the identity provider potentially federates the Principal's identity at the identity provider with the Principal's identity at the service provider.

Note:

Under certain conditions, an identity provider may unilaterally (without receiving an authentication request) issue an authentication response to a service provider.

The identity provider may be proxying for an *authenticating identity provider*, in which case, this protocol may be repeated between the recipient of the original `<AuthnRequest>`, and other identity providers (see Section 3.2.2.7).

3.2.1. Request

The service provider issues an initial `<AuthnRequest>` to an identity provider. A set of parameters is included in the request that allows the service provider to specify desired behavior at identity providers in processing the request. A requester can control the following identity provider behaviors:

- Prompt the Principal for credentials if the Principal is not presently authenticated.

- Prompt the Principal for credentials, even if the Principal is presently authenticated.

- Federate the Principal's identity at the identity provider with the Principal's identity at the requester.

- Issue an anonymous and temporary identifier for the Principal to the service provider.

- Use a specific protocol profile in responding to the request.

- Use a specific authentication context (for example, smartcard-based authentication vs. username/password-based authentication).

- Restrict the ability of the recipient to proxy the authentication request to additional identity providers.

Additionally, the service provider MAY include any desired state information in the request that the identity provider should relay back to the service provider in the response.

The `<AuthnRequest>` message SHOULD be signed. If the requesting provider's `<AuthnRequestsSigned>` metadata element is "true", then any request messages it generates MUST be signed.

3.2.1.1. Element <AuthnRequest>

The `<AuthnRequest>` is defined as an extension of `samlp:RequestAbstractType`. The `RequestID` attribute in **`samlp:RequestAbstractType`** has uniqueness requirements placed on it by [SAMLCore11], which require it to have the properties of a nonce.

The elements of the request are as follows:

`Extension` [Optional]
: Optional container for protocol extensions established by agreement between providers. Implementers should note that this element may not contain content from the core Liberty namespace (which is prevented at the schema level by `requiring namespace="##other"`).

`ProviderID` [Required]
: The requester's unique identifier.

`AffiliationID` [Optional]
: If present, indicates that the requester is acting as a member of the affiliation group identified.

`NameIDPolicy` [Optional]
: An enumeration permitting requester influence over name identifier policy at the identity provider.

`IsPassive` [Optional]
: If "true," specifies that the identity provider MUST NOT interact with the Principal and MUST NOT take control of the user interface from the service provider. If "false," the identity provider MAY interact with the user and MAY temporarily take control of the user interface for that purpose. If not specified, "true" is presumed.

`ForceAuthn` [Optional]
: Controls whether the identity provider authenticates the Principal regardless of whether the Principal is already authenticated. This element is specified only when `<IsPassive>` is "false." If `<ForceAuthn>` is "true," specifies that the identity provider MUST always authenticate the Principal, regardless of whether the Principal is presently authenticated. If "false," specifies that the identity provider MUST re-authenticate the user only if the Principal is not presently authenticated. If not specified, "false" is presumed.

`ProtocolProfile` [Optional]
: The protocol profile that the requester wishes to use for the response. If the element is not specified, the default protocol profile is *http://projectliberty.org/profiles/brws-art*, defined in [LibertyBindProf].

`AssertionConsumerServiceID` [Optional]
: Used to direct the identity provider to use a specific assertion consumer service URL at the service provider. It references an element in the provider's metadata with a matching id attribute.

`RequestAuthnContext` [Optional]
: Information describing which authentication context the requester desires the identity provider to use in authenticating the Principal.

`RelayState` [Optional]
: This contains state information that will be relayed back in the response. This data SHOULD be integrity-protected by the request author and MAY have other protections placed on it by the request author. An

example of such protection is confidentiality. Note that the actual requested resource URL at the service provider SHOULD NOT be directly placed in this element unless the attendant privacy considerations do not apply.

`Scoping` [Optional]
Specifies any preferences on the number and specific identifiers of additional identity providers through which the authentication request may be proxied. See Section 3.2.2.7 for rules regarding the proxying of identity providers.
The requester may also choose not to include this element, in which case, the recipient of the message MAY act as a proxy, according to the rules in Section 3.2.2.7

`consent` [Optional]
Indicates whether or not consent has been obtained from a user in sending this message.

The `<RequestAuthnContext>` element may contain the following elements, the first two of which are mutually exclusive:

`AuthnContextClassRef` [Optional]
The ordered set of authentication context class references the requester desires the identity provider to use in authenticating the Principal.

`AuthnContextStatementRef` [Optional]
The ordered set of exact authentication statements the requester desires the identity provider to use in authenticating the Principal.

`AuthnContextComparison` [Optional]
If set to "exact", then the identity provider is asked to match at least one of the specified `<AuthnContext>` elements exactly. This can also be set to "minimum", which asks that the identity provider use a context that he feels is at least as good as any specified in the `<AuthnContext>`, "better", which means that the they can use any context better than any that were supplied, or "maximum", which means to use a context that is at most as strong as any specified. If not specified, this is assumed to be "exact".

The `<Scoping>` element may contain the following elements:

`ProxyCount` [Optional]
The upper limit on the number of proxying steps the requester wishes to specify for this authentication request.

`IDPList` [Optional]
An ordered list of identity providers which the requester prefers to use in authenticating the Principal. This list is a suggestion only, and may be ignored or added to by the recipient of the message.

The schema fragment defining the element and its type is as follows:

```
<xs:element name="AuthnRequest" type="AuthnRequestType"/>
<xs:complexType name="AuthnRequestType">
   <xs:complexContent>
      <xs:extension base="samlp:RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="ProviderID"/>
            <xs:element ref="AffiliationID" minOccurs="0"/>
            <xs:element ref="NameIDPolicy" minOccurs="0"/>
            <xs:element name="ForceAuthn" type="xs:boolean" minOccurs="0"/>
            <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
            <xs:element ref="ProtocolProfile" minOccurs="0"/>
            <xs:element name="AssertionConsumerServiceID" type="xs:string" minOccurs="0"/>
```

```
            <xs:element ref="RequestAuthnContext" minOccurs="0"/>
            <xs:element ref="RelayState" minOccurs="0"/>
            <xs:element ref="Scoping" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute ref="consent" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:simpleType name="NameIDPolicyType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="none"/>
      <xs:enumeration value="onetime"/>
      <xs:enumeration value="federated"/>
      <xs:enumeration value="any"/>
   </xs:restriction>
</xs:simpleType>
<xs:element name="NameIDPolicy" type="NameIDPolicyType"/>
<xs:simpleType name="AuthnContextComparisonType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="exact"/>
      <xs:enumeration value="minimum"/>
      <xs:enumeration value="maximum"/>
      <xs:enumeration value="better"/>
   </xs:restriction>
</xs:simpleType>
<xs:complexType name="ScopingType">
   <xs:sequence>
      <xs:element name="ProxyCount" type="xs:nonNegativeInteger" minOccurs="0"/>
      <xs:element ref="IDPList" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:element name="Scoping" type="ScopingType"/>
<xs:element name="RelayState" type="xs:string"/>
<xs:element name="ProtocolProfile" type="xs:anyURI"/>
<xs:element name="RequestAuthnContext">
   <xs:complexType>
      <xs:sequence>
         <xs:choice>
            <xs:element name="AuthnContextClassRef" type="xs:anyURI" maxOccurs="unbounded"/>
            <xs:element name="AuthnContextStatementRef" type="xs:anyURI" maxOccurs="unbounded"/>
         </xs:choice>
         <xs:element name="AuthnContextComparison" type="AuthnContextComparisonType"
minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

3.2.1.2. Example

```
            <lib:AuthnRequest RequestID="RPCUk2ll+GVz+t1lLURp51oFvJXk"
            MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
            IssueInstant="2001-12-17T21:42:4Z" xmlns:lib="urn:liberty:iff:2003-08">
            <ds:Signature> ... </ds:Signature>
            <lib:ProviderID>http://ServiceProvider.com</lib:ProviderID>
            <lib:NameIDPolicy>federate</lib:NameIDPolicy>
            <lib:ForceAuthn>false</lib:ForceAuthn>
            <lib:IsPassive>false</lib:IsPassive>
            <lib:ProtocolProfile>http://projectliberty.org/profiles/brws-post</lib:ProtocolProfile>
            <lib:RequestAuthnContext>
            <lib:AuthnContextClassRef>http://projectliberty.org/schemas/
            authctx/classes/Password-ProtectedTransport</lib:AuthnContextClassRef>
            <lib:AuthnContextComparison>exact</lib:AuthnContextComparison>
```

```
            </lib:RequestAuthnContext>
               <lib:RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</lib:RelayState>
               <lib:Scoping>
               <lib:ProxyCount>1</lib:ProxyCount>
            </lib:Scoping>
            </lib:AuthnRequest>
```

3.2.2. Response

The response is either an `<AuthnResponse>` element containing a set of authentication assertions or a set of artifacts the service provider can dereference into a set of authentication assertions.

All authentication assertions generated by an identity provider for a service provider MUST be of type **`AssertionType`**. The `<Subject>` element in any subject statement MUST be of type **`SubjectType`**.

The `<Subject>` element MUST contain `<saml:SubjectConfirmation>`, with at least one `<saml:ConfirmationMethod>` in accordance with the SSO profile in use (see [LibertyBindProf]). There MAY be additional `<saml:ConfirmationMethod>` elements that go beyond the requirements of the SSO profile.

If the `<NameIDPolicy>` element is omitted or "none" and assuming a federation exists for the principal between the identity provider and service provider, then if the service provider registered a name identifier for the Principal, the `<saml:NameIdentifier>` element in the `<saml:Subject>` element MUST be the service provider-provided name identifier for the Principal. Otherwise, `<saml:NameIdentifier>` MUST be the most current name identifier supplied by the identity provider. The `<IDPProvidedNameIdentifier>` MUST contain the most recent name identifier supplied by the identity provider. In either case, the `Format` attribute MUST either be *urn:liberty:iff:nameid:federated*, or in the case of a non-1.2 identity federation may contain a non-standard `Format` value.

If the `<AffiliationID>` element is present, then the `<saml:NameIdentifier>` MUST be the most recent name identifier provided by a member of the affiliation, if any, or the name identifier for the Principal supplied by the identity provider for the affiliation.

If the `<NameIDPolicy>` element is *onetime*, then the `<saml:NameIdentifier>` element in the `<saml:Subject>` element MUST be a temporary, one-time-use identifier for the Principal, with a `Format` attribute of *urn:liberty:iff:nameid:one-time*.

If the `<NameIDPolicy>` element is *federated*, then a new identity federation MAY be created, if one does not already exist for the Principal and policy permits. The response is then constructed as if the value were *none*.

If the `<NameIDPolicy>` element is *any*, then evaluation proceeds as if the value were *federated*. If the policy for the Principal forbids federation, then evaluation MAY proceed as if the value were *onetime*.

All authentication statements MUST be of type **`AuthenticationStatementType`**.

Identity providers MUST include a `<saml:AudienceRestrictionCondition>` element that specifies the intended consumers of the assertion. One `<saml:Audience>` element MUST be set to the intended recipient's `ProviderID`. The recipient MUST validate that it is the intended viewer before using the assertion. The assertion MAY contain additional `<saml:Audience>` elements that specify other intended relying parties.

Identity providers MUST include a `SessionIndex` attribute in resulting authentication statements, which is used to aid the identity provider in managing multiple sessions with the Principal. Subsequent messages from the service provider to the identity provider that are session-dependent MUST include this `SessionIndex` attribute.

Identity providers MAY include other types of statements in the assertion(s) returned, depending on agreements between providers and other specifications that provide additional functionality. Any such statements that include a name identifier representing the Principal MUST be consistent with the identification semantics dictated by the

<NameIDPolicy> element. This is particularly relevant if the "onetime" policy is in effect; a temporary identifier or an otherwise obfuscated and protected value MUST be used.

Each assertion in the <AuthnResponse> message MUST be individually signed by the identity provider (that is, each assertion must contain a Signature element which signs only the assertion). It is RECOMMENDED that the signature be omitted from the <AuthnResponse> itself, but signing of the message is not forbidden.

3.2.2.1. Element <AuthnResponse>

The type AuthnResponseType is extended from samlp:ResponseType.

The response contains the following elements:

Extension [Optional]
: Optional container for protocol extensions established by agreement between providers.

ProviderID [Required]
: The identity provider's unique identifier.

RelayState [Optional]
: This contains state information being relayed.

consent [Optional]
: Indicates whether or not consent has been obtained from a user in sending this message.

The schema fragment is as follows:

```
<xs:element name="AuthnResponse" type="AuthnResponseType"/>
<xs:complexType name="AuthnResponseType">
   <xs:complexContent>
      <xs:extension base="samlp:ResponseType">
         <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="ProviderID"/>
            <xs:element ref="RelayState" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute ref="consent" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

3.2.2.2. Element <AssertionType>

Authentication assertions provided in an <AuthnResponse> element MUST be of type **AssertionType**, which is an extension of **saml:AssertionType**, so that the RequestID attribute from the original <AuthnRequest> MAY be included in the InResponseTo attribute in the <Assertion> element. This is done because it is not required that the <AuthnResponse> element itself be signed. Instead, the individual <Assertion> elements contained MUST each be signed.

Note that it is optional for the InResponseTo to be present. Its absence indicates that the <AuthnResponse> has been unilaterally sent by the identity provider without a corresponding <AuthnRequest> message from the service provider. If the attribute is present, it MUST be set to the RequestID of the original <AuthnRequest>.

The schema fragment is as follows:

```
<xs:element name="Assertion" type="AssertionType"/>
<xs:complexType name="AssertionType">
```

```
      <xs:complexContent>
         <xs:extension base="saml:AssertionType">
            <xs:attribute name="InResponseTo" type="xs:NCName" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

3.2.2.3. SubjectType and Related Types

The type **`SubjectType`**, extended from **`saml:SubjectType`**, is used to include the optional `<IDPProvidedNameIdentifier>` element in subject statements.

Liberty name identifier elements are directly based on **`saml:NameIdentifierType`**. Liberty ID-FF use of SAML-defined attributes is as follows - it should be noted that `NameQualifier` and `Format` attributes are more specifically defined in Liberty messages than in SAML (see [SAMLCore11]) and their omission has specific meaning:

`Format` [Required Except for Pre-1.2 Federations]
Indicates the format, semantics, and processing rules associated with the identifier. If this attribute is omitted, or if a value other than those listed below is used, then the value *urn:liberty:iff:nameid:federated* is effectively assumed, but no other 1.2 semantics are to be implied, including use of `NameQualifier` as described below. That is, such an identifier is assumed to be a pre-1.2 federated identifier and the `NameQualifier` attribute, if present at all, is to be treated as literal data. Otherwise, for federations established between 1.2 providers, one of these four values MUST be present:

urn:liberty:iff:nameid:federated
Used for identifiers communicated on behalf of Principals that have federated their identity.

urn:liberty:iff:nameid:one-time
Used for identifiers with anonymous, single-use semantics communicated on behalf of Principals that have not federated or wish to act anonymously.

urn:liberty:iff:nameid:encrypted
Used for identifiers that have been encrypted for use only by a specific provider. Such an identifier MUST be a base-64 encoded `<EncryptedNameIdentifier>` element, defined below. See the Name Identifier Encryption Profile in [LibertyBindProf] for more information.

urn:liberty:iff:nameid:entityID
Used for identifiers that identify a Liberty provider or affiliation group. This may be used to indicate that the subject indicated in an authentication statement is a Liberty provider or affiliation group.

`NameQualifier` [Optional]
Generally used to indicate the unique identifier of the service provider or affiliation group for or by whom the name identifier was created. Unless otherwise specified, the service provider's or affiliation group's unique identifier MUST be placed in this attribute to disambiguate the identifier from any other identity federations the principal may have with the identity provider.
An issuer MAY omit this attribute if the containing element is in the subject of a statement in an assertion that contains a single `<saml:Audience>` equal to the unique identifier of the value that this attribute would have had. This typically means that an assertion issued during SSO to a service provider that is not acting as part of an affiliation group can omit this attribute.
The above interpretation of this attribute applies ONLY if the `Format` attribute contains one of the four values listed above. Otherwise, the identifier MUST be assumed to represent a pre-1.2 identity federation and MUST be treated only as literal data.

The **`EncryptedNameIdentifierType`** is a container for an encrypted XML element and wrapped encryption key, for use when encrypting name identifiers. It contains the following elements:

`xenc:EncryptedData` [Required]
The result of applying XML encryption to an `<EncryptableNameIdentifier>` element. The `Type` attribute of this element should be set to *http://www.w3.org/2001/04/xmlenc#Element*, per [xmlenc-core]. This element MAY contain other elements permitted by [xmlenc-core] that indicate the key to be used in decrypting the identifier, if that key has been exchanged out of band via some other mechanism agreed to by the providers.

`xenc:EncryptedKey` [Optional]
A wrapped symmetric encryption key used to encrypt the `<EncryptableNameIdentifier>` element, per [xmlenc-core].

`EncryptableNameIdentifierType` is an extension of **`saml:NameIdentifierType`** that contains additional attributes designed to help receiving providers restrict reuse of encrypted identifiers. Use of existing SAML attributes and definitions of new attributes for the `<EncryptableNameIdentifier>` element follows:

`Format` [Required]
MUST contain a value of *urn:liberty:iff:nameid:federated* (of the formats defined in this specification, only federated name identifiers sometimes require encryption), excepting that Pre-1.2 federated identifiers being encrypted MAY omit this attribute or use another unspecified value, as described earlier.

`NameQualifier` [Required]
MUST contain the URI-based identifier of the provider or affiliation group that is encrypting the Principal's federated name identifier, so that the receiving provider can determine which identity federation is involved. This is often an identity provider, and is thus an exception to the general rule that a service provider's identifier be used.

`IssueInstant` [Required]
MUST contain the date and time at which the encrypted element is produced. Receiving providers SHOULD use this value to prevent long periods of time elapsing between creation and use of encrypted identifiers, in accordance with the profile of use.

`Nonce` [Optional]
MAY contain a unique pseudorandom value. The probability of two randomly chosen values being identical MUST be less than or equal to 2^{-128} and SHOULD be less than or equal to 2^{-160}. Receiving providers MAY use this value to prevent unacceptable reuse of the same encrypted identifier.

The schema fragment is as follows:

```
<xs:complexType name="SubjectType">
   <xs:complexContent>
      <xs:extension base="saml:SubjectType">
         <xs:sequence>
            <xs:element ref="IDPProvidedNameIdentifier" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:element name="Subject" type="SubjectType" substitutionGroup="saml:Subject"/>
<xs:element name="EncryptableNameIdentifier" type="EncryptableNameIdentifierType"
   substitutionGroup="saml:NameIdentifier"/>
<xs:complexType name="EncryptableNameIdentifierType">
   <xs:simpleContent>
      <xs:extension base="saml:NameIdentifierType">
         <xs:attribute name="IssueInstant" type="xs:dateTime"/>
         <xs:attribute name="Nonce" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
<xs:element name="EncryptedNameIdentifier" type="EncryptedNameIdentifierType"/>
```

```
   <xs:complexType name="EncryptedNameIdentifierType">
      <xs:sequence>
            <xs:element ref="xenc:EncryptedData"/>
            <xs:element ref="xenc:EncryptedKey" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
```

3.2.2.4. Type AuthenticationStatementType

The type **`AuthenticationStatementType`** is an extension of **`saml:AuthenticationStatementType`**, which allows for the following elements and attributes:

`AuthnContext` [Optional]
: The context used by the identity provider in the authentication event that yielded this statement. Contains either an authentication context statement or a reference to an authentication context statement. Optionally contains a reference to an authentication context class.

`ReauthenticateOnOrAfter` [Optional]
: The time at, or after which the service provider reauthenticates the Principal with the identity provider (as required in the Section 3.2.2.6 [24]processing rules for this protocol).

`SessionIndex` [Required]
: Indexes the particular session between the Principal and the identity provider under which this authentication statement is being issued. This value SHOULD be a small, positive integer but may be any string of text. However, this value MUST NOT be a globally unique value for the Principal's session at the identity provider.

When an `<AuthnContext>` element is specified, the `saml:AuthenticationMethod` attribute on the `<saml:AuthenticationStatement>` MUST be *urn:liberty:ac:2003-08*.

When the service provider is processing a `<saml:AuthenticationStatement>` of type **`lib:AuthenticationStatementType`** and the `saml:AuthenticationMethod` attribute is *urn:liberty:ac:2003-08*, the service provider MUST refer to the `<AuthnContext>` element and ignore the `saml:AuthenticationMethod` attribute.

The schema fragment is as follows:

```
   <xs:element name="AuthenticationStatement" type="AuthenticationStatementType"
substitutionGroup="saml:Statement"/>
   <xs:complexType name="AuthenticationStatementType">
      <xs:complexContent>
         <xs:extension base="saml:AuthenticationStatementType">
            <xs:sequence>
               <xs:element name="AuthnContext" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="AuthnContextClassRef" type="xs:anyURI" minOccurs="0"/>
                        <xs:choice>
                           <xs:element ref="ac:AuthenticationContextStatement"/>
                           <xs:element name="AuthnContextStatementRef" type="xs:anyURI"/>
                        </xs:choice>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
            <xs:attribute name="ReauthenticateOnOrAfter" type="xs:dateTime" use="optional"/>
            <xs:attribute name="SessionIndex" type="xs:string" use="required"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

3.2.2.5. Example

```
                <lib:AuthnResponse ResponseID="hhuuja1bc744hGJn5Q9A5yvEIgS"
                   InResponseTo="Zon3WjJ2KL7j+bJu7MuIr4Pt2go5" MajorVersion="1" MinorVersion="2"
                   consent="urn:liberty:consent:obtained" IssueInstant="2002-10-31T21:55:41Z">
                   <samlp:Status>
                      <samlp:StatusCode Value="samlp:Success"/>
                   </samlp:Status>
                   <lib:Assertion MajorVersion="1" MinorVersion="2"
                      AssertionID="e06e5a28-bc80-4ba6-9ecb-712949db686e"
                      Issuer="http://IdentityProvider.com" IssueInstant="2001-12-17T09:30:47Z"
                      InResponseTo="4e7c3772-4fa4-4a0f-99e8-7d719ff6067c">
                      <saml:Conditions NotBefore="2001-12-17T09:30:47Z"
NotOnOrAfter="2001-12-17T09:35:47Z">
                         <saml:AudienceRestrictionCondition>
                            <saml:Audience>http://ServiceProvider.com</saml:Audience>
                         </saml:AudienceRestrictionCondition>
                      </saml:Conditions>
                      <lib:AuthenticationStatement AuthenticationInstant="2001-12-17T09:30:47Z"
                         SessionIndex="3" ReauthenticateOnOrAfter="2001-12-17T11:30:47Z"
                         AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
                         <lib:Subject>
                            <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
                               Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-
4c68-be35-cc9e7db39e2b</saml:NameIdentifier>
                            <saml:SubjectConfirmation>
                               <saml:ConfirmationMethod>urn:oasis:names:tc:SAM
L:1.0:cm:bearer</saml:ConfirmationMethod>
                            </saml:SubjectConfirmation>
                            <IDPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
                               Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-
4c68-be35-cc9e7db39e2b</IDPProvidedNameIdentifier>
                         </lib:Subject>
                      </lib:AuthenticationStatement>
                      <ds:Signature>...</ds:Signature>
                   </lib:Assertion>
                   <lib:ProviderID>http://IdentityProvider.com</lib:ProviderID>
                   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
                </lib:AuthnResponse>
```

3.2.2.6. Processing Rules

Generally, an identity provider will receive an authentication request, and process that request in order to generate an authentication response.

It is possible for an identity provider to generate an authentication response without first having received an authentication request (see [LibertyImplGuide] for more information).

When an identity provider initiates SSO without first receiving an authentication request, it MUST generate the response using the processing rules specified below, as if it had received an authentication request with the following (minimal) content. Furthermore, the resulting assertion(s) MUST NOT include an `InResponseTo` attribute.

A `<ProviderID>` and optionally an `<AffiliationID>` that represent the service provider (possibly acting as an affiliation group) to whom the response will be sent.

A `<NameIDPolicy>` of *any*.

Optionally, a <RelayState> with a value understood by mutual agreement between the identity provider and service provider so that the service provider knows how to handle subsequent interactions with the Principal after SSO. This MAY be the URL of a resource at the service provider.

When an identity provider receives an authentication request, it MUST process the request according to the following rules:

The <ProviderID> in the request MAY be the ProviderID of a known provider with which the identity provider has established a relationship, or out of band means MAY be used to establish such a relationship.

The <ProviderID> MUST be resolvable to at least one (default) assertion consumer service URL at the requesting provider that the identity provider may use when returning the corresponding assertion reference.

The following rules apply to choosing the appropriate URL to use:

If the <AssertionConsumerServiceID> element is provided, then the identity provider MUST search for the value among the id attributes in the <AssertionConsumerServiceURL> elements in the provider's metadata to determine the URL to use. If no match can be found, then the provider MUST return an error with a second-level <samlp:StatusCode> of *lib:InvalidAssertionConsumerServiceIndex* to the default URL (the <AssertionConsumerServiceURL> with an isDefault attribute of "true").

If the <AssertionConsumerServiceID> element is not provided, then the identity provider MUST use the default URL (the <AssertionConsumerServiceURL> with an isDefault attribute of "true").

<ds:Signature>, if present, MUST be the signature of the service provider as specified by the <ProviderID>.

If the requesting provider's <AuthnRequestsSigned> metadata element is "true", then any request messages it generates MUST be signed. If an unsigned request is received, then the provider MUST return an error with a second-level <samlp:StatusCode> of *lib:UnsignedAuthnRequest*.

If <IsPassive> is "true," the identity provider MUST NOT interact with the Principal and MUST NOT take control of the user interface (if applicable).

The identity provider MUST attempt to authenticate the Principal if <ForceAuthn> is "true," regardless of whether the Principal is presently authenticated, unless <IsPassive> is "true."

Success in authenticating the Principal is indicated by a status code of *samlp:Success* and a signed assertion containing at least one statement of type **lib:AuthenticationStatementType** representing the Principal's authentication information. Other types of statements may also be included, as defined by providers and other specifications.

Failure to authenticate the Principal is indicated by a status code other than *samlp:Success.*. For failures, assertions MUST NOT be included in the <AuthnResponse>.

<AffiliationID>, if present, MUST be the unique identifier of a known affiliation group with which the identity provider has an established relationship, and of which the requesting provider is a member. If present, identity providers MUST establish and resolve federations based on the specified affiliation, not the requesting provider. In addition, identity providers MAY retrieve information regarding the other members of the affiliation group by querying metadata (see [LibertyMetadata]) and present a list of members of the affiliation group to the Principal.

The following rules apply to the selection of name identifiers and the federation process:

If the <NameIDPolicy> element is omitted or "none", then the identity provider MUST return the name identifier(s) corresponding to the federation that exists between the identity provider and the requesting provider or affiliation group for the Principal. If no such federation exists, then an error with a second-level <samlp:StatusCode> of *lib:FederationDoesNotExist* MUST be returned to the provider.

If the <NameIDPolicy> element is "onetime", then the <saml:NameIdentifier> element in the <saml:Subject> element MUST be a temporary, one-time-use identifier for the Principal, with a Format attribute of *urn:liberty:iff:nameid:one-time*.

If <NameIDPolicy> is *federated*, and if the Principal consents, then the identity provider MAY federate the Principal's identity with the requesting provider (or the affiliation group if <AffiliationID> is present). If the identity provider already has a previous federation on record for the Principal's identity at the requesting provider or affiliation group (such as when a provider previously issued a <FederationTerminationNotification> which was not received by the identity provider), then the identity provider SHOULD treat the request as if <NameIDPolicy> were *none*.

If <NameIDPolicy> is *any*, then the rules above for the values of *federated* and *onetime* MUST be followed, in that order. Thus, a new federation may be created, an existing federation used, or a temporary identifier generated.

When including a Principal's federated identity in the response, the <Subject> element MUST include a <saml:NameIdentifier>. If the service provider or affiliation group has set its own identifier for the Principal, then the <saml:NameIdentifier> MUST be set to that value. The identifier set by the identity provider MUST then be included in the <IDPProvidedNameIdentifier>.

If a one-time identifier is being used, or if the service provider or affiliation group has not set its own identifier for the Principal, then the <saml:NameIdentifier> MUST contain the identifier set by the identity provider for the Principal. The <IDPProvidedNameIdentifier> SHOULD be omitted in this case, since it would contain identical information.

The <Subject> MUST contain a <SubjectConfirmation> element in accordance with the SSO profile used to return the response (see [SAMLBind11] and [LibertyBindProf]).

When federating or in the case of a temporary value, the identity provider MUST adhere to the following rules in generating the name identifier:

The name identifier MUST be unique across all Principals in the scope of that requester-identity provider relationship.

The name identifier for the specific Principal MUST be unique across all providers with which an identity federation exists with the identity provider.

The identity provider MUST respond using the specified <ProtocolProfile>.

If <RelayState> contains a value, the identity provider MUST include this value in unmodified form in the <RelayState> element of the returned authentication assertion.

The *InResponseTo* attribute in all generated <Assertion> elements in the <AuthnResponse> element MUST be set to the value of the RequestID attribute in the corresponding <AuthnRequest> element. If there is no such request because the identity provider is initiating the response on its own, then the attribute MUST NOT be included.

Additionally, if the <RequestAuthnContext> element is specified, the identity provider MUST authenticate the Principal according to the following rules:

If one or more <AuthnContextClassRef> or <AuthnContextStatementRef> elements are included, 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 an authentication statement in the resulting assertion.

Additionally, if an <AuthnContextComparison> element is supplied, and one or more <AuthnContextStatementRef> or <AuthnContextClassRef> elements are included, then the resulting authentication statement in the assertion (if any) MUST follow the rule specified in the <AuthnContextComparison> element. If this requirement cannot be satisfied, the identity provider MUST NOT include an authentication statement in the resulting assertion.

If <AuthnContextComparison> 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 <AuthnContextComparison> 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 identity provider) as one of the authentication contexts specified.

If <AuthnContextComparison> 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 <AuthnContextComparison> 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.

If the identity provider wishes to rely on a second identity provider as the source of the Principal's authentication, then the provider MUST follow the rules specified in Section 3.2.2.7.

If the requesting provider attempts to federate a Principal's identity with an identity provider, but another Principal's identity at the same requesting provider is already federated with the same identity provider, it will receive the other Principal's established name identifier in the <AuthnResponse>, rather than a new random one. The requesting provider MUST detect this error and handle it appropriately without leaving either Principal's identity at the provider in an unusable state.

The resulting authentication statement in the assertion by the identity provider MAY contain a ReauthenticateOnOrAfter attribute. If this attribute is included, the service provider MUST send a new <AuthnRequest> for the Principal to the identity provider at the next point of interaction with the Principal on or after the time specified by the ReauthenticateOnOrAfter attribute. It is then up to the identity provider to authenticate the user.

Note: The Principal may already have an authenticated session with the identity provider, in which case the identity provider should generate a new authentication assertion without any intervention by the Principal.

3.2.2.7. Dynamic Proxying of Identity Providers

An identity provider that is asked to authenticate a known Principal that it believes has already authenticated to another identity provider may make an authentication request on behalf of the requesting provider to that authenticating identity provider.

The originator of an authentication request may control proxy behavior by including a <Scoping> element where the provider sets a desired <ProxyCount value and/or indicates a list of preferred identity providers which may be proxied by defining an ordered <IDPList> of preferred providers.

The identity provider MUST conform to the following processing rules when choosing to proxy an authentication request:

The identity provider MAY proxy an authentication request if the value in the <ProxyCount> element is greater than zero, or if no <ProxyCount> appears in the request. Whether it chooses to proxy or not is a matter of local policy.

The identity provider MAY choose to proxy for a provider specified in the <IDPList> but is not required to do so.

The identity provider MUST NOT proxy a request where the <ProxyCount> is set to zero.

If the <ProxyCount> element has a value of zero, then the identity provider MUST return an error containing a second-, or lower-level <samlp:StatusCode> value of *lib:ProxyCountExceeded*, unless it can directly authenticate the principal.

When creating the new authentication request:

The identity provider MUST include equivalent or stricter forms of all the information included in the original authentication request (such as authentication context policy).

If the authenticating provider is not a Liberty provider that implements this specifications, then the proxying provider MUST have some other way to ensure that the elements governing Principal interaction (`<IsPassive>`, for example) will be honored by the authenticating provider.

The new request MUST contain a `<ProxyCount>` element with a value of at least one less than the original value. If the original request does not contain a `<ProxyCount>` element, then the new request SHOULD contain a `<ProxyCount>` element.

If an `<IDPList>` was specified in the original request, the new request MUST also contain an `<IDPList>`.

The identity provider MAY add additional identity providers to the end of `<IDPList>`, but MUST NOT remove providers from the list.

The authentication request and response are processed in normal fashion, in accordance with the rules given in Section 3.2. Once the Principal has authenticated to the proxying identity provider, the following steps are followed:

The proxying identity provider prepares a new authentication assertion on its own behalf by copying in the relevant information from the original assertion. The original assertion will be restricted by `AudienceRestrictionCondition` to (at least) the identity provider, while the new assertion's condition will reference (at least) the original requesting provider.

If the `<NameIdentifier>` has one-time semantics (determined by examining the `Format`), then the identity provider MUST generate a new one-time identifier and include it in the new assertion.

If the `<NameIdentifier>` is not one-time, then the identity provider MUST include the Principal's federated `<IDPProvidedNameIdentifier>` for the requesting provider or affiliation group, as well as the `<NameIdentifier>` provided by the requesting provider or affiliation group, if any.

If the identity provider does rely on a second provider to authenticate the principal, then its response to the original requester MUST include an `<AuthnContext>` element containing an `<ac:AuthenticatingAuthority>` element referencing the identity provider to which the responding provider referred the Principal.

If the original assertion contains `<AuthnContext>` information that includes one or more `<ac:AuthenticatingAuthority>` elements, those elements SHOULD be included in the new assertion, with the new element placed after them.

Any other `<AuthnContext>` information MAY be copied, translated, or omitted in accordance with the policies of the identity provider, provided that the original requirements dictated by the requesting provider are met.

If the authenticating identity provider is not a Liberty provider that implements the ID-FF specifications, then the proxying identity provider MUST generate a `ProviderID` value for the authenticating provider. This value SHOULD be consistent over time across different requests. The value MUST not conflict with values used or generated by other Liberty providers.

If, in the future the identity provider is asked to authenticate the same Principal for a second provider, and this provider's request is equally or less strict than the original provider's request, the identity provider MAY skip the creation of a new request to the authenticating identity provider. The concrete definition of "less strict" and "equivalent" is up to the identity provider, following the guidelines in section 3.2.3.

3.2.2.8. Active Intermediaries

In some profiles, an intermediary is active between the service provider's authentication request and the identity provider's authentication response. Examples of an active intermediary include a user agent or client proxy that implements the "Liberty-Enabled Client and Proxy Profile" described in [LibertyBindProf].

NOTE: an active intermediary has the capability to return status codes to the service provider it interacts with. For example, the intermediary may be unable to contact an identity provider identified by the service provider, and the intermediary may return a status code to the service provider indicating that an error occurred. Status codes MUST be conveyed within `<AuthnResponse>` messages using the `<samlp:Status>` element, according to the rules specified in [SAMLCore11], utilizing second-, and lower-level `<samlp:StatusCode>` elements. Specific values are defined below. Service providers should also note that intermediaries are not providers, and hence may not have clocks as accurately synchronized. This may invalidate the `IssueInstant` attribute included in the `<AuthnResponse>` received by the service provider.

For all profiles specifying an active intermediary, the profile specification must:

Specify whether the `<AuthnRequest>` element sent from the service provider to the identity provider via the intermediary is wrapped in an `<AuthnRequestEnvelope>`. See Section 3.2.3.

Specify whether the `<AuthnResponse>` element sent from the identity provider to the service provider via the intermediary is wrapped in an `<AuthnResponseEnvelope>`. See Section 3.2.4.

3.2.2.8.1. Processing Rules for Active Intermediaries

For all profiles specifying an active intermediary, the intermediary MUST follow these processing rules:

If the profile specifies that the message sent from the service provider to the identity provider, via the intermediary, is wrapped in an `<AuthnRequestEnvelope>`:

The intermediary MUST remove the enveloping `<AuthnRequestEnvelope>` before forwarding the `<AuthnRequest>` element to the identity provider.

The intermediary MAY locally generate `<AuthnResponse>` elements and send them to the service provider using the `<AssertionConsumerServiceURL>` contained within the `<AuthnRequestEnvelope>`. Such `<AuthnResponse>` elements MUST NOT contain any `<lib:Assertion>` elements. The `<AuthnResponse>` elements MUST have an `InResponseTo` attribute set to the RequestID of the `<AuthnRequest>` that could not be serviced, if any. If the `<AuthnRequest>` contained a `<RelayState>` element, the `<AuthnResponse>` MUST include a `<RelayState>` element with its value set to that supplied in the `<AuthnRequest>`. Such responses MAY be generated as a result of local errors on the intermediary, and MAY indicate the underlying reasons in the `<samlp:Status>` element in the `<AuthnResponse>`.

If the profile specifies that the message from the identity provider to the service provider, via the intermediary, is wrapped in an `<AuthnResponseEnvelope>`

- The intermediary MUST remove the enveloping `<AuthnResponseEnvelope>` before forwarding the `<AuthnResponse>` element to the service provider.

- The intermediary MUST send `<AuthnResponse>` messages received from the identity provider to the service provider using the `<AssertionConsumerServiceURL>` contained within the `<AuthnResponseEnvelope>` sent by the identity provider.

3.2.2.9. Status Code Values for Error Conditions

If an error occurs in the processing at an identity provider or an intermediary, the following values are defined for use in second-, or lower-level nested `<samlp:StatusCode>` elements, if the responder wishes to provide additional detail. If reporting specific status values will not expose the responder or the Principal to security risk or exposure of unnecessary information, then as much detail as possible SHOULD be returned.

lib:FederationDoesNotExist: Used by an identity provider to indicate that the Principal has not federated his or her identity with the service provider, and the service provider indicated a requirement for federation.

lib:UnknownPrincipal: Used by an identity provider to indicate that the Principal is not known to it.

lib:NoAuthnContext: Used by an identity provider to indicate that the specified authentication context information in the request prohibits authentication from taking place.

lib:NoPassive: Used by an identity provider or an intermediary to indicate that authentication of the Principal requires interaction and cannot be performed passively.

lib:ProxyCountExceeded: Used by an identity provider to indicate that it cannot authenticate the principal itself, and was not permitted to relay the request further.

lib:NoAvailableIDP: Used by an intermediary to indicate that none of the supported identity provider URLs from the `<IDPList>` can be resolved or that none of the supported identity providers are available.

lib:NoSupportedIDP: Used by an intermediary to indicate that none of the identity providers are supported by the intermediary.

3.2.3. Request Envelope

Some profiles MAY wrap the `<AuthnRequest>` element in an envelope. This envelope allows for extra processing by an intermediary between the service provider and the identity provider. An example of an intermediary is a user agent or proxy. Processing rules are given in section 3.2.3.3.1. Note that the envelope is for consumption by the intermediary and is removed before the enveloped `<AuthnRequest>` element is forwarded to the identity provider.

To facilitate the removal of the envelope by the intermediary, the service provider MUST ensure that the XML obtained by removing the `<AuthnRequestEnvelope>` from the enclosed `<AuthnRequest>` is well-formed and valid.

3.2.3.1. Element <AuthnRequestEnvelope>

The authentication request envelope contains the following elements:

`Extension` [Optional]
: Optional container for protocol extensions established by agreement between service providers and intermediaries. Implementers should note that this element may not contain content from the core Liberty namespace (which is prevented at the schema level by requiring namespace="##other").

`AuthnRequest` [Required]
: The authentication request contained within the envelope.

`ProviderID` [Required]
: The requester's ProviderID.

`ProviderName` [Optional]
: The human-readable name of the requester.

`AssertionConsumerServiceURL` [Required]
: A URL specifying where `<AuthnResponse>` elements, locally generated by an intermediary, should be sent. See the processing rules for active intermediaries specified in section 3.2.3.1.1.

`IDPList` [Optional]
: A list of identity providers, from which, one may be chosen to service the authentication request.

`IsPassive` [Optional]
If "true," specifies that any intermediary between the service provider and identity provider MUST NOT interact with the Principal. If not specified, "true" is presumed.

The schema fragment is as follows:

```
<xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType"/>
<xs:complexType name="AuthnRequestEnvelopeType">
   <xs:complexContent>
      <xs:extension base="RequestEnvelopeType">
         <xs:sequence>
            <xs:element ref="AuthnRequest"/>
            <xs:element ref="ProviderID"/>
            <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
            <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
            <xs:element ref="IDPList" minOccurs="0"/>
            <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="RequestEnvelopeType">
   <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

3.2.3.2. Element <IDPList>

In the request envelope, some profiles may wish to allow the service provider to transport a list of identity providers to the user agent. This specification provides a schema that profiles SHOULD use for this purpose. The elements are as follows:

`IDPList` [Optional]
The container element for an IDP List.

`IDPEntries` [Required]
Contains a list of identity provider entries.

`IDPEntry` [Required]
Describes an identity provider that the service provider supports.

`ProviderID` [Required]
The identity provider's unique identifier.

`ProviderName` [Optional]
The identity provider's human-readable name.

`Loc` [Optional]
The identity provider's URI, to which authentication requests may be sent. If present, this MUST be set to the value of the identity provider's `<SingleSignOnService>` element, obtained from their metadata ([LibertyMetadata]).

`GetComplete` [Optional]
If the identity provider list is not complete, this element may be included with a URI that points to where the complete list can be retrieved.

The schema fragment is as follows:

```
   <xs:element name="IDPList" type="IDPListType"/>
   <xs:complexType name="IDPListType">
      <xs:sequence>
         <xs:element ref="IDPEntries"/>
         <xs:element ref="GetComplete" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="IDPEntry">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="ProviderID"/>
            <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
            <xs:element name="Loc" type="xs:anyURI"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="IDPEntries">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="IDPEntry" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="GetComplete" type="xs:anyURI"/>
```

3.2.3.3. Example

```
                  <AuthnRequestEnvelope>
                  <AuthnRequest> ... </AuthnRequest>
                  <ProviderID>http://ServiceProvider.com</Provider ID>
                  <ProviderName>Service Provider X</ProviderName>
                  <AssertionConsumerServiceURL>http://ServiceProvi der.com/lecp_assertion_consume
r</AssertionConsumerServiceURL >
                  <IDPList>
                  <IDPEntries>
                  <IDPEntry>
                  <ProviderID>http://IdentityProvider.com</Provid erID>
                  <ProviderName>Identity Provider X</ProviderName>
                  <Loc>http://www.IdentityProvider.com/liberty/s so</Loc>
               </IDPEntry>
               </IDPEntries>
                  <GetComplete>https://ServiceProvider.com/idplist?i d=604be136-fe91-441e-afb8-f887
48ae3b8b </GetComplete>
               </IDPList>
                  <IsPassive>0</IsPassive>
               </AuthnRequestEnvelope>
```

3.2.4. Response Envelope

As with the `<AuthnRequest>` element, some profiles MAY wrap the `<AuthnResponse>` element in an envelope. This envelope allows for extra processing by an intermediary (such as a user agent or proxy) between the identity provider and the service provider. Applicable processing rules are given in section 3.2.3.3.1. Note that the envelope is for consumption by the intermediary and is removed prior to the forwarding of the enveloped `<AuthnResponse>` element to the service provider.

3.2.4.1. Element <AuthnResponseEnvelope>

The authentication response envelope contains the following elements:

`Extension` [Optional]
Optional container for protocol extensions established by agreement between service providers and intermediaries. Implementers should note that this element may not contain content from the core Liberty namespace (which is prevented at the schema level by requiring namespace="##other").

`AuthnResponse` [Required]
The enveloped authentication response.

`AssertionConsumerServiceURL` [Required]
The service provider's URL where the authentication response should be sent. This element's value SHOULD be obtained from the element of the same name in the service provider's metadata (see [LibertyMetadata]).

The schema fragment is as follows:

```
<xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType"/>
<xs:complexType name="AuthnResponseEnvelopeType">
   <xs:complexContent>
      <xs:extension base="ResponseEnvelopeType">
         <xs:sequence>
            <xs:element ref="AuthnResponse"/>
            <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="ResponseEnvelopeType">
   <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

3.2.4.2. Example

```
            <AuthnResponseEnvelope>
            <AuthnResponse> ... </AuthnResponse>
            <AssertionConsumerServiceURL>
            http://ServiceProvider.com/lecp_assertion_consumer
          </AssertionConsumerServiceURL>
          </AuthnResponseEnvelope>
```

3.3. Name Registration Protocol

During federation, the identity provider generates an opaque value that serves as the initial name identifier that both the service provider and the identity provider use in referring to the Principal when communicating with each other. This name identifier is termed the `<IDPProvidedNameIdentifier>`.

Subsequent to federation, the service provider MAY register a different opaque value with the identity provider. This opaque value is termed the `<SPProvidedNameIdentifier>`. Until the service provider registers a different name, the identity provider will use `<IDPProvidedNameIdentifier>` to refer to the Principal when communicating with the service provider.

After a service provider's name registration, the identity provider MUST use the `<SPProvidedNameIdentifier>` for `<saml:NameIdentifier>` elements when communicating to the service provider about the Principal. The service provider MUST use the current (most recently supplied) `<IDPProvidedNameIdentifier>` for `<saml:NameIdentifier>` elements when communicating to the identity provider about the Principal.

Either the service provider or the identity provider MAY register a new name identifier for a Principal with each other at any time following federation. The name identifiers specified by providers SHOULD adhere to the following guidelines:

- The name identifier SHOULD be unique across the identity providers with which the Principal's identity is federated.
- The name identifier SHOULD be unique within the group of name identifiers that have been registered with the identity provider by this service provider.
- Only federated identifiers (and by extension pre-1.2 identifiers which are always federated) can be replaced and set with this protocol. Encrypted or one-time identifiers MUST NOT be used.
- The `Format` attribute in the newly registered identifier element MUST be `urn:liberty:iff:nameid:federated`. The `Format` attribute in the `<OldProvidedNameIdentifier>` element MUST either be `urn:liberty:iff:nameid:federated`, or MAY be omitted or set to a non-standard value if a pre-1.2 federation is being updated. This also applies to the `<SPProvidedNameIdentifier>` sent by an IdP or the `<IDPProvidedNameIdentifier>` sent by an SP if referencing a pre-1.2 federation.
- Similarly, the `NameQualifier` attribute in the newly registered identifier MUST be the service provider's unique identifier, an affiliation group's unique identifier, or omitted (implying the service provider's identifier). The other elements either follow the same rules, or if a pre-1.2 federation, MUST be treated as literal data.

3.3.1. Request

To register a `<SPProvidedNameIdentifier>` with an identity provider, the service provider sends a `<RegisterNameIdentifierRequest>` message.

The same `<RegisterNameIdentifierRequest>` message may be sent by an identity provider, seeking to change the `<IDPProvidedNameIdentifier>` stored by the service provider.

The `<RegisterNameIdentifierRequest>` message SHOULD be signed.

3.3.1.1. Element <RegisterNameIdentifierRequest>

The elements of the message are as follows:

`Extension` [Optional]
Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]
The provider's identifier.

`IDPProvidedNameIdentifier` [Required]
For an IdP-initiated request, the new name identifier the service provider should use when communicating with the identity provider. For an SP-initiated request, the current name identifier established by the IdP for the SP to use when communicating with it.

`SPProvidedNameIdentifier` [Optional]
For an SP-initiated request, this is required, and is the new name identifier the identity provider should use when communicating to the service provider. For an IdP- initiated request, this is the current name identifier established by the SP for the IdP to use when communicating with it, or if none exists, MUST be omitted.

`OldProvidedNameIdentifier` [Required]
In the case of either provider choosing to request a change of provided name identifiers, this element holds the previous version set by that provider. For a service provider making their first name change following federation, the `<OldProvidedNameIdentifier>` will contain the current `<IDPProvidedNameIdentifier>`.

`RelayState` [Optional]
This contains state information that will be relayed back in the response. This data SHOULD be integrity-protected by the request author and MAY have other protections placed on it by the request author. An example of such protection is confidentiality.

The schema fragment is as follows:

```
   <xs:element name="RegisterNameIdentifierRequest" type="RegisterNameIdentifierRequestType"/>
   <xs:complexType name="RegisterNameIdentifierRequestType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="IDPProvidedNameIdentifier"/>
               <xs:element ref="SPProvidedNameIdentifier" minOccurs="0"/>
               <xs:element ref="OldProvidedNameIdentifier"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
   <xs:element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
   <xs:element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
```

3.3.1.2. Example

```
                <RegisterNameIdentifierRequest RequestID="eb20e77f-d982-44f9-936e-dd135bf437d4"
                   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">
                   <ds:Signature>...</ds:Signature>
                   <ProviderID>http://ServiceProvider.com</ProviderID>
                   <IDPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
                      Format="urn:liberty:iff:nameid:federated">342ad3d8-93e
e-4c68-be35-cc9e7db39e2b</IDPProvidedNameIdentifier>
                   <SPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
                      Format="urn:liberty:iff:nameid:federated">e958019a</S
PProvidedNameIdentifier>
```

```
                    <OldProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
                       Format="urn:liberty:iff:nameid:federated">e8950
14a</OldProvidedNameIdentifier>
                    <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
                 </RegisterNameIdentifierRequest>
```

3.3.2. Response

The recipient MUST respond with a `<RegisterNameIdentifierResponse>` message, which is of type **`StatusResponseType`**. **`StatusResponseType`** is an extension of **`samlp:ResponseType`** and a `<samlp:Status>` element and a `<RelayState>` may exist in the body.

This message SHOULD be signed.

3.3.2.1. Element <RegisterNameIdentifierResponse>

The elements of the message are as follows:

`Extension` [Optional]
: Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]
: The provider's unique identifier.

`Status` [Required]
: The status of the request processing.

`RelayState` [Optional]
: This element contains state information that will be relayed back in the response, if it has been supplied in the request.

The schema fragment is as follows:

```
   <xs:element name="RegisterNameIdentifierResponse" type="StatusResponseType"/>
   <xs:complexType name="StatusResponseType">
      <xs:complexContent>
         <xs:extension base="samlp:ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="samlp:Status"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

3.3.2.2. Example

```
                 <RegisterNameIdentifierResponse ResponseID="ff74ec0f-1165-4fa3-b0
88-3dd2c2388b91"
                    InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
                    MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
                    Recipient="http://ServiceProvider.com">
                    <ds:Signature>...</ds:Signature>
                    <ProviderID>http://ServiceProvider.com</ProviderID>
                    <samlp:Status>
                       <samlp:StatusCode Value="samlp:Success"/>
```

```
                    </samlp:Status>
                    <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
                </RegisterNameIdentifierResponse>
```

3.3.3. Processing Rules

The recipient MUST validate any signature present on the message. To be considered valid, the signature provided MUST be the signature of the `<ProviderID>` contained in the message.

If the request includes an `<IDPProvidedNameIdentifier>` for which no federation exists between the service provider and the identity provider, the identity provider MUST respond with a `<samlp:Status>` element containing a second-level `<samlp:StatusCode>` of lib:FederationDoesNotExist. Otherwise, the identity provider MUST use `<SPProvidedNameIdentifier>` when subsequently communicating to the service provider regarding this Principal.

Either provider MAY choose to change their provided name identifier. In this case, the `<OldProvidedNameIdentifier>` should contain the previous version of their name identifier. When a service provider chooses to change their provided name identifier, the `<OldProvidedNameIdentifier>` should contain the current `<SPProvidedNameIdentifier>`. Note that when they first change their name, this will be equal to the `<IDPProvidedNameIdentifier>`. Similarly, when an identity provider wishes to change their provided name identifier, they will move the previous version to the `<OldProvidedNameIdentifier>` when sending this message.

In all of the name identifier elements in the request and response messages of this protocol, if the Principal's identity federation is between the identity provider and an affiliation group in which the service provider is a member, then the `NameQualifier` attribute MUST contain the unique identifier of the affiliation group. Otherwise, it MUST contain the unique identifier of the service provider or MAY be omitted (implying the service provider's identifier). The exception as noted previously is if the old identifier(s) represent a pre-1.2 federation being updated with the request, in which case the attribute is simply treated as literal data or omitted.

Changes to these identifiers may take a potentially significant amount of time to propagate through the systems at both the sender and the receiver. Implementations may wish to allow each party to accept either identifier for some period of time following the successful completion of a name identifier change. Not doing so could result in the inability of the Principal to access resources.

If `<RelayState>` contains a value, the recipient MUST include this value in unmodified form in the `<RelayState>` element of the response.

3.4. Federation Termination Notification Protocol

When the Principal terminates an identity federation between a service provider and an identity provider from the service provider, the service provider MUST send a `<FederationTerminationNotification>` message to the identity provider. The service provider is stating that it will no longer accept authentication assertions from the identity provider for the specified Principal.

Likewise, when the Principal terminates an identity federation from the identity provider, the identity provider MUST send a `<FederationTerminationNotification>` message to the service provider. In this case, the identity provider is stating that it will no longer provide authentication assertions to the service provider for the specified Principal.

This notification message is a one-way asynchronous message. Reasonable, best-effort delivery MUST be employed by all providers sending this message.

3.4.1. Message

The provider sends a `<FederationTerminationNotification>` to the provider with which it is terminating a federation.

The <FederationTerminationNotification> message SHOULD be signed.

3.4.1.1. Element <FederationTerminationNotification>

The elements are as follows:

`Extension` [Optional]
Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]
The identifier of the provider that is sending this message.

`NameIdentifier` [Required]
The name identifier of the Principal terminating federation. This name identifier MUST be equal to the <saml:NameIdentifier> element (and its included attributes) agreed upon earlier between the two communicating providers.

`consent` [Optional]
Indicates whether or not consent has been obtained from a user in sending this message.

The schema fragment is as follows:

```
   <xs:element name="FederationTerminationNotification" type="FederationTerminationNotif
icationType"/>
   <xs:complexType name="FederationTerminationNotificationType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="saml:NameIdentifier"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

3.4.1.2. Example

```
<FederationTerminationNotification RequestID="e9c2-eb65-4bce-ab8f-4becdf229815"
   MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
   IssueInstant="2001-12-17T09:30:47Z">
   <ds:Signature>...</ds:Signature>
   <ProviderID>http://IdentityProvider.com</ProviderID>
   <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
      Format="urn:liberty:iff:nameid:federated">e958019a</saml:NameIdentifier>
   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</FederationTerminationNotification>
```

3.4.2. Processing Rules

The receiving provider MUST validate any signature present on the message. The signature on the message MUST be the signature of the <ProviderID> contained in the message. If the signature is not valid, the provider MUST ignore the message.

If a provider receives a federation termination notification message that refers to a federation that does not exist from the perspective of the provider, the provider MUST ignore the message. Otherwise, the provider MAY perform any maintenance with the knowledge that the federation has been terminated.

A provider MAY choose to invalidate the session of a user for whom federation has been terminated.

If the Principal's identity federation was between the identity provider and an affiliation group in which the service provider is a member, then the `NameQualifier` attribute MUST contain the unique identifier of the affiliation group. Otherwise, it MUST contain the unique identifier of the service provider or MAY be omitted (implying the service provider's identifier). The exception is a pre-1.2 federated identifier (recognizable by a missing or non-standard `Format` attribute), in which case the `NameQualifier` attribute is simply treated as literal data or omitted.

3.5. Single Logout Protocol

The Single Logout Protocol provides a message exchange protocol by which all sessions authenticated by a particular identity provider are near-simultaneously terminated. The Single Logout Protocol is used either when a Principal logs out at a service provider or when the Principal logs out at an identity provider.

The Principal may have established authenticated sessions both with the identity provider, and individual service providers, based on authentication assertions supplied by the identity provider.

When the Principal invokes the single logout process at a service provider, the service provider MUST send a `<LogoutRequest>` message to the identity provider that provided the authentication service related to that session at the service provider.

When either the Principal invokes a logout at the identity provider or a service provider sends a logout request to the identity provider specifying that Principal, the identity provider MUST send a `<LogoutRequest>` message to each service provider to which it provided authentication assertions under its current session with the Principal, with the exception of the service provider that sent the `<LogoutRequest>` message to the identity provider.

The following rules apply to identity providers involved in authentication proxying:

- If the identity provider is proxying authentication from a second identity provider, then it MUST send a `<LogoutRequest>` to the proxied identity provider, unless the proxying provider has already received a `<LogoutRequest>` from the proxied provider.

- If the identity provider has provided authentication assertions on behalf of a Principal to a proxying identity provider, then it MUST send a `<LogoutRequest>` to that provider, unless the proxying provider has already received a `<LogoutRequest>` from the proxied provider.

3.5.1. Request

The <LogoutRequest> message indicates to the message receiver that a Principal's session was terminated. The message includes an optional SessionIndex element that MUST be specified if and only if a SessionIndex element was present in the authentication statement in the assertion that the service provider used in establishing the session with the Principal. This message SHOULD be signed.

3.5.1.1. Element <LogoutRequest>

Extension [Optional]
: Optional container for protocol extensions established by agreement between providers. Implementers should note that this element may not contain content from the core Liberty namespace (which is prevented at the schema level by requiring namespace="##other").

NameIdentifier [Required]
: The name identifier of the Principal that logged out. This name identifier MUST be equal to the <saml:NameIdentifier> element (including the equality of contained attributes) agreed upon between the two communicating providers.

ProviderID [Required]
: The identifier of the provider that is making the request.

SessionIndex [Optional]
: The session index specified in the authentication statement of the assertion used to establish the session being terminated. If a <SessionIndex> element was present in the authentication statement, an identical <SessionIndex> MUST be present in the <LogoutRequest>. If no <SessionIndex> element was present in the authentication statement, the <SessionIndex> MUST be omitted from the <LogoutRequest>.

RelayState [Optional]
: This may contain state information that will be relayed back in the response. This data SHOULD be integrity-protected by the request author and MAY have other protections placed on it by the request author. An example of such protection is confidentiality.

consent [Optional]
: Indicates whether or not consent has been obtained from a user in sending this message.

NotOnOrAfter [Optional]
: This attribute is used to specify the time instant at which the recipient may expire a logout request from the sender. The sender MAY set this value equal to the NotOnOrAfter attribute of a previously-supplied assertion, if one was specified.

The schema fragment is as follows:

```
   <xs:element name="LogoutRequest" type="LogoutRequestType"/>
   <xs:complexType name="LogoutRequestType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="saml:NameIdentifier"/>
               <xs:element name="SessionIndex" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
            <xs:attribute name="NotOnOrAfter" type="xs:dateTime" use="optional"/>
```

```
        </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

3.5.1.2. Example

```
                <LogoutRequest RequestID="d4769303-7c33-4d65-931f-ddeb19fa6a73"
                   MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
                   IssueInstant="2001-12-17T09:30:47Z">
                   <ds:Signature>...</ds:Signature>
                   <ProviderID>http://ServiceProvider.com</ProviderID>
                   <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
                      Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be3
5-cc9e7db39e2b</saml:NameIdentifier>
                   <SessionIndex>3</SessionIndex>
                   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
                </LogoutRequest>
```

3.5.2. Response

The recipient MUST return a `<LogoutResponse>` message, which is of type `StatusResponseType`.

This message SHOULD be signed.

3.5.2.1. Element <LogoutResponse>

The elements of the message are as follows:

`Extension` [Optional]
: Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]
: The identifier of the provider responding.

`Status` [Required]
: A status code that indicates the result of the request.

`RelayState` [Optional]
: This contains state information that may have appeared in the request, and is being relayed back to the sender.

The schema fragment is as follows:

```
   <xs:element name="LogoutResponse" type="StatusResponseType"/>
```

3.5.2.2. Example

```
                <LogoutResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
                   InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
                   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
                   Recipient="http://ServiceProvider.com">
                   <ds:Signature>...</ds:Signature>
                   <ProviderID>http://IdentityProvider.com</ProviderID>
                   <samlp:Status>
                      <samlp:StatusCode Value="samlp:Success"/>
                   </samlp:Status>
```

```
                    <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZt uMFQxDS8b</RelayState>
                </LogoutResponse>
```

3.5.2.3. Processing Rules

If `<RelayState>` contains a value, the recipient MUST include this value in unmodified form in the `<RelayState>` element of the response.

If the Principal's identity federation was between the identity provider and an affiliation group in which the service provider is a member, then the `NameQualifier` attribute MUST contain the unique identifier of the affiliation group. Otherwise, it MUST contain the unique identifier of the service provider or MAY be omitted (implying the service provider's identifier). The exception is a pre-1.2 federated identifier (recognizable by a missing or non-standard `Format` attribute), in which case the `NameQualifier` attribute is simply treated as literal data or omitted.

Other unique processing rules apply based on whether the message receiver is an identity provider or a service provider.

3.5.2.3.1. Identity Provider Processing Rules

When an identity provider receives the `<LogoutRequest>` message, the identity provider MUST validate that any signature present on the message is the signature of a service provider to which the identity provider provided an authentication assertion for the current session. If that holds, the identity provider SHOULD do the following:

- Send a `<LogoutRequest>` message to each service provider for which the identity provider provided authentication assertions in the current session, other than the originator of the `<LogoutRequest>`.
- Send a `<LogoutRequest>` message to the identity provider on behalf of whom the identity provider proxied the user's authentication, unless the second identity provider is the originator of the `<LogoutRequest>`.
- Terminate the Principal's current session as specified by the `<saml:NameIdentifier>` element, and any `<SessionIndex>` present in the logout request message.

When constructing a logout request message, the identity provider MUST set the value of the *NotOnOrAfter* attribute of the message to a time value, indicating an expiration time for the message. See [LibertyImplGuide] for more details.

If an error occurs during this further processing of the logout (for example, relying service providers may not all implement the Single Logout profile used by the requesting service provider), then the identity provider MUST respond to the original requester with a `<LogoutResponse>` message, indicating the status of the logout request. The value "lib:UnsupportedProfile" is provided for a second-level `<samlp:StatusCode>`, indicating that a service provider should retry the `<LogoutRequest>` using a different profile.

3.5.2.3.2. Service Provider Processing Rules

When the service provider receives the `<LogoutRequest>` message, the service provider MUST validate the identity provider's signature contained in the `<ds:Signature>` element. If the signature is that of the identity provider that provided the authentication for the Principal's current session, the service provider MUST invalidate the Principal's session(s) referred to by the `<saml:NameIdentifier>` element, and any `SessionIndex` elements supplied in the message.

The service provider MUST apply the logout request message to any assertion that meets the following conditions, *even if the assertion arrives after the logout request*:

1. The `SessionIndex` of the assertion matches one specified in the logout request.
2. The assertion would otherwise be valid

3. The logout request has not yet expired (determined by examining the `NotOnOrAfter` attribute on the message).

When constructing a logout request message, the service provider SHOULD omit the *NotOnOrAfter* attribute.

3.6. Name Identifier Mapping Protocol

When a service provider requires a name identifier for a Principal with which it has an identity federation relationship, but which references an identity federation between the identity provider and another service provider, it can use this protocol to obtain such an identifier. This allows the requesting provider to communicate with the other service provider about the Principal without an identity federation for the Principal between them. The resulting value SHOULD be encrypted so as to obscure the actual value from anyone but the second service provider. To the requester, it will be an opaque (and one-time) value.

Upon receipt of a `<NameIdentifierMappingRequest>` message, an identity provider that supports this protocol MUST respond with a `<NameIdentifierMappingResponse>` message.

3.6.1. Request

The requesting service provider sends a `<NameIdentifierMappingRequest>` message to the identity provider which can provide the desired federated name identifier.

The `<NameIdentifierMappingRequest>` message MUST be signed.

3.6.1.1. Element <NameIdentifierMappingRequest>

The elements and attributes are as follows:

`Extension` [Optional]
: Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]
: The unique identifier of the provider that is sending this message.

`saml:NameIdentifier` [Required]
: The name identifier of the Principal for whom the requester is obtaining a mapped identifier. See the processing rules below for additional information on the content of this element.

`TargetNamespace` [Required]
: The unique identifier of the service provider or affiliation group for whom the requester needs the name identifier of the Principal to subsequently communicate with.

`consent` [Optional]
: Indicates whether or not consent has been obtained from a user in sending this message.

The schema fragment is as follows:

```
<xs:element name="NameIdentifierMappingRequest" type="NameIdentifierMappingRequestTyp e"/>
<xs:complexType name="NameIdentifierMappingRequest Type">
   <xs:complexContent>
      <xs:extension base="samlp:RequestAbstractType">
         <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="ProviderID"/>
            <xs:element ref="saml:NameIdentifier"/>
            <xs:element name="TargetNamespace" type="md:entityIDType"/>
         </xs:sequence>
         <xs:attribute ref="consent" use="optional"/>
      </xs:extension>
```

```
      </xs:complexContent>
   </xs:complexType>
```

3.6.1.2. Example

```
                <NameIdentifierMappingRequest RequestID="e9c2-eb65-4bce-ab8f-4bec df229815"
                   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">
                   <ds:Signature>...</ds:Signature>
                   <ProviderID>http://RequestingServiceP rovider.com</ProviderID>
                   <saml:NameIdentifier NameQualifier="http://RequestingServiceProvi der.com"
                      Format="urn:liberty:iff:nameid:federated">e95 8019a</saml:NameIdentifier>
                   <TargetNamespace>http://TargetSe rviceProvider.com</TargetNames pace>
                </NameIdentifierMappingRequest >
```

3.6.2. Response

The responding provider MUST return a `<NameIdentifierMappingResponse>` message if the signature is valid.

This message SHOULD be signed.

3.6.2.1. Element <NameIdentifierMappingResponse>

The elements of the message are as follows:

`Extension` [Optional]
: Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]
: The identifier of the provider responding.

`Status` [Required]
: A status code that indicates the reception and processing status of the message.

`saml:NameIdentifier` [Optional]
: If the request is successful, contains the resulting mapped name identifier for the desired identity federation, which SHOULD be in encrypted form. See the processing rules below for additional information on the content of this element.

The schema fragment is as follows:

```
   <xs:element name="NameIdentifierMappingResponse" type="NameIdentifierMappingResponseT ype"/>
   <xs:complexType name="NameIdentifierMappingRespo nseType">
      <xs:complexContent>
         <xs:extension base="samlp:ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="samlp:Status"/>
               <xs:element ref="saml:NameIdentifier" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

3.6.2.2. Example

```
                <NameIdentifierMappingResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
                   InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
                   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
                   Recipient="http://ServiceProvider.com">
                   <ds:Signature>...</ds:Signature>
                   <ProviderID>http://IdentityProvider.com</ProviderID>
                   <samlp:Status>
                      <samlp:StatusCode Value="samlp:Success"/>
                   </samlp:Status>
                   <saml:NameIdentifier NameQualifier="http://TargetServiceProvider.com"
                      Format="urn:liberty:iff:nameid:federated">gfhfrfe89u43</saml:Name
Identifier>
                </NameIdentifierMappingResponse>
```

3.6.3. Processing Rules

The receiving provider MUST validate any signature present on the message. The signature on the message MUST be the signature of the `<ProviderID>` contained in the message. If the signature is not valid, the provider MUST ignore the message.

In the request message's `<saml:NameIdentifier>`, if the Principal's identity federation is between the identity provider and an affiliation group in which the requesting service provider is a member, then the `NameQualifier` attribute MUST contain the unique identifier of the affiliation group. Otherwise, it MUST contain the unique identifier of the service provider.

3.6.3.1. Recipient Processing Rules

Supporting identity providers MUST respond with a `<NameIdentifierMappingResponse>` message.

If the identity provider is unable to identify the Principal referenced in the request, then it MUST return a response message with a second-level `<samlp:StatusCode> Value` of *lib:UnknownPrincipal*.

If the identity provider is unable to map the name identifier of the Principal to a federation between it and the `<TargetNamespace>` in the request, then it MUST return a response message with a second-level `<samlp:StatusCode> Value` of *lib:FederationDoesNotExist*.

If the mapping is successful, then the response message's `<saml:NameIdentifier>` MUST contain a `NameQualifier` that matches the request's `<TargetNamespace>` element. It SHOULD be a a one-time encrypted value but MAY be the actual federated identifier value; the `Format` attribute MUST indicate this distinction.

4. Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:iff:2003-08"
   xmlns="urn:liberty:iff:2003-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:ac="urn:liberty:ac:2003-08"
   xmlns:md="urn:liberty:metadata:2003-08"
   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   elementFormDefault="qualified" attributeFormDefault="unqualified">

   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
      schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
      schemaLocation="oasis-sstc-saml-schema-protocol-1.1.xsd"/>
   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
   <xs:import namespace="urn:liberty:ac:2003-
08" schemaLocation="liberty-authentication-context-v1.3.xsd"/>
   <xs:import namespace="urn:liberty:metadata:2003-0
8" schemaLocation="liberty-metadata-v1.1.xsd"/>

   <xs:include schemaLocation="liberty-idff-utility-v1.0.xsd"/>

   <xs:annotation>
      <xs:documentation> ### IMPORTANT NOTICE ###

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

Liberty Protocols and Schema Specification
Version 1.2-errata-v3.0
19 May 2005
Updated for WSF 1.1 release

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

</xs:documentation>
   </xs:annotation>
   <xs:element name="ProviderID" type="md:entityIDType"/>
   <xs:element name="AffiliationID" type="md:entityIDType"/>

   <xs:element name="AuthnRequest" type="AuthnRequestType"/>
   <xs:complexType name="AuthnRequestType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="AffiliationID" minOccurs="0"/>
               <xs:element ref="NameIDPolicy" minOccurs="0"/>
               <xs:element name="ForceAuthn" type="xs:boolean" minOccurs="0"/>
               <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
               <xs:element ref="ProtocolProfile" minOccurs="0"/>
               <xs:element name="AssertionConsumerServiceID" type="xs:string" minOccurs="0"/>
               <xs:element ref="RequestAuthnContext" minOccurs="0"/>
               <xs:element ref="RelayState" minOccurs="0"/>
               <xs:element ref="Scoping" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:simpleType name="NameIDPolicyType">
      <xs:restriction base="xs:string">
```

```
         <xs:enumeration value="none"/>
         <xs:enumeration value="onetime"/>
         <xs:enumeration value="federated"/>
         <xs:enumeration value="any"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:element name="NameIDPolicy" type="NameIDPolicyType"/>
   <xs:simpleType name="AuthnContextComparisonType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="exact"/>
         <xs:enumeration value="minimum"/>
         <xs:enumeration value="maximum"/>
         <xs:enumeration value="better"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="ScopingType">
      <xs:sequence>
         <xs:element name="ProxyCount" type="xs:nonNegativeInteger" minOccurs="0"/>
         <xs:element ref="IDPList" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="Scoping" type="ScopingType"/>
   <xs:element name="RelayState" type="xs:string"/>
   <xs:element name="ProtocolProfile" type="xs:anyURI"/>
   <xs:element name="RequestAuthnContext">
      <xs:complexType>
         <xs:sequence>
            <xs:choice>
               <xs:element name="AuthnContextClassRef" type="xs:anyURI" maxOccurs="unbounded"/>
               <xs:element name="AuthnContextStatementRef" type="xs:anyURI" maxOccurs="unbounded"/>
            </xs:choice>
            <xs:element name="AuthnContextComparison" type="AuthnContextComparisonType"
minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="AuthnResponse" type="AuthnResponseType"/>
   <xs:complexType name="AuthnResponseType">
      <xs:complexContent>
         <xs:extension base="samlp:ResponseType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="Assertion" type="AssertionType"/>
   <xs:complexType name="AssertionType">
      <xs:complexContent>
         <xs:extension base="saml:AssertionType">
            <xs:attribute name="InResponseTo" type="xs:NCName" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="SubjectType">
      <xs:complexContent>
         <xs:extension base="saml:SubjectType">
            <xs:sequence>
               <xs:element ref="IDPProvidedNameIdentifier" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

```
   <xs:element name="Subject" type="SubjectType" substitutionGroup="saml:Subject"/>
   <xs:element name="EncryptableNameIdentifier" type="EncryptableNameIdentifierType"
      substitutionGroup="saml:NameIdentifier"/>
   <xs:complexType name="EncryptableNameIdentifierType">
      <xs:simpleContent>
         <xs:extension base="saml:NameIdentifierType">
            <xs:attribute name="IssueInstant" type="xs:dateTime"/>
            <xs:attribute name="Nonce" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:element name="EncryptedNameIdentifier" type="EncryptedNameIdentifierType"/>
   <xs:complexType name="EncryptedNameIdentifierType">
      <xs:sequence>
           <xs:element ref="xenc:EncryptedData"/>
           <xs:element ref="xenc:EncryptedKey" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>

   <xs:element name="AuthenticationStatement" type="AuthenticationStatementType"
substitutionGroup="saml:Statement"/>
   <xs:complexType name="AuthenticationStatementType">
      <xs:complexContent>
         <xs:extension base="saml:AuthenticationStatementType">
            <xs:sequence>
               <xs:element name="AuthnContext" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="AuthnContextClassRef" type="xs:anyURI" minOccurs="0"/>
                        <xs:choice>
                           <xs:element ref="ac:AuthenticationContextStatement"/>
                           <xs:element name="AuthnContextStatementRef" type="xs:anyURI"/>
                        </xs:choice>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
            <xs:attribute name="ReauthenticateOnOrAfter" type="xs:dateTime" use="optional"/>
            <xs:attribute name="SessionIndex" type="xs:string" use="required"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType"/>
   <xs:complexType name="AuthnRequestEnvelopeType">
      <xs:complexContent>
         <xs:extension base="RequestEnvelopeType">
            <xs:sequence>
               <xs:element ref="AuthnRequest"/>
               <xs:element ref="ProviderID"/>
               <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
               <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
               <xs:element ref="IDPList" minOccurs="0"/>
               <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="RequestEnvelopeType">
      <xs:sequence>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="IDPList" type="IDPListType"/>
   <xs:complexType name="IDPListType">
      <xs:sequence>
         <xs:element ref="IDPEntries"/>
         <xs:element ref="GetComplete" minOccurs="0"/>
```

```
      </xs:sequence>
   </xs:complexType>
   <xs:element name="IDPEntry">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="ProviderID"/>
            <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
            <xs:element name="Loc" type="xs:anyURI"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="IDPEntries">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="IDPEntry" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="GetComplete" type="xs:anyURI"/>
   <xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType"/>
   <xs:complexType name="AuthnResponseEnvelopeType">
      <xs:complexContent>
         <xs:extension base="ResponseEnvelopeType">
            <xs:sequence>
               <xs:element ref="AuthnResponse"/>
               <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:complexType name="ResponseEnvelopeType">
      <xs:sequence>
         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="RegisterNameIdentifierRequest" type="RegisterNameIdentifierRequestType"/>
   <xs:complexType name="RegisterNameIdentifierRequestType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="IDPProvidedNameIdentifier"/>
               <xs:element ref="SPProvidedNameIdentifier" minOccurs="0"/>
               <xs:element ref="OldProvidedNameIdentifier"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
   <xs:element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
   <xs:element name="OldProvidedNameIdentifier " type="saml:NameIdentifierType"/>
   <xs:element name="RegisterNameIdentifierResponse" type="StatusResponseType"/>
   <xs:complexType name="StatusResponseType">
      <xs:complexContent>
         <xs:extension base="samlp:ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="samlp:Status"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="FederationTerminationNotification
```

```
" type="FederationTerminationNotificationType"/>
   <xs:complexType name="FederationTerminationNotificationType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="saml:NameIdentifier"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="LogoutRequest" type="LogoutRequestType"/>
   <xs:complexType name="LogoutRequestType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="saml:NameIdentifier"/>
               <xs:element name="SessionIndex" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
               <xs:element ref="RelayState" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
            <xs:attribute name="NotOnOrAfter" type="xs:dateTime" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="LogoutResponse" type="StatusResponseType"/>
   <xs:element name="NameIdentifierMappingRequest" type="NameIdentifierMappingRequestType"/>
   <xs:complexType name="NameIdentifierMappingRequestType">
      <xs:complexContent>
         <xs:extension base="samlp:RequestAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="saml:NameIdentifier"/>
               <xs:element name="TargetNamespace" type="md:entityIDType"/>
            </xs:sequence>
            <xs:attribute ref="consent" use="optional"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="NameIdentifierMappingResponse" type="NameIdentifierMappingResponseType"/>
   <xs:complexType name="NameIdentifierMappingResponseType">
      <xs:complexContent>
         <xs:extension base="samlp:ResponseAbstractType">
            <xs:sequence>
               <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element ref="ProviderID"/>
               <xs:element ref="samlp:Status"/>
               <xs:element ref="saml:NameIdentifier" minOccurs="0"/>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

References

Normative

[LibertyBindProf] Cantor, Scott, Kemp, John, Champagne, Darryl, eds. "Liberty ID-FF Bindings and Profiles Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

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

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

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

[RFC3280] Housley, R., eds. (April 2002). "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3280.txt [April 2002]*.

[SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[SAMLBind11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

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

[Schema2] Biron, Paul V., Malhotra, Ashok, eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-2/*

[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*

[XMLCanon] Boyer, John, Eastlake, Donald, Reagle, Joseph, eds. (18 July 2002). "Exclusive XML Canonicalization," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xml-exc-c14n*

[xmlenc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing," W3C Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlenc-core/*

Informative

[LibertyImplGuide] "Liberty ID-FF Implementation Guidelines," Version 1.2, Liberty Alliance Project (18 April 2004). *http://www.projectliberty.org/specs* Thompson, Peter, Champagne, Darryl, eds.