Liberty ID-FF Protocols and Schema Specification

Version 1.2 – 08

11 April 2003

Editors:

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

Contributors:

Robert Aarts, Nokia
Xavier Serret, Gemplus
Greg Whitehead, Trustgenix

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 Core specification.

Notice

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

51 # History

Revision	Date	Log
01	10-Feb-03	Addition of ResourceStatementType for ISF; Removal of provider metadata
02	18-Feb-03	Changes for working with multiple IdPs ; Revised type for ResourceStatementType
03	10-Mar-03	Affiliations text changes, second round of changes for IFF 1.2 from Houston feedback
04	20-Mar-03	Responses to feedback and bugzilla issues
05	28-Mar-03	Added text to note that an IdP may prompt the Principal with a list of affiliated providers
06	04-Apr-03	Reformatted title page, code, example, reference sections ; 3.1.3 Allowed affiliationID to be set to the providerID of one of the member providers, changed 'the' URI to 'any' URI ; 3.1.11 Added reference to metadata ; 3.2.3.2 Restricted proxy count resending to values > 0 ; 3.2.3.2 919 - made SHOULD from should ; 3.5 Attempted to clarify language on proxying of providers ; 3.6 Attempted to clarify introductions ; 3.7 Attempted to clarify termination of introductions
07	11-Apr-03	Added consent attribute to AuthnRequest, FedTermNot, LogoutRequest messages + text

52

Table of 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:1.2`). 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:1.2`)

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

The prefix `samlp:` stands for the SAML 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 `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 [LibertyGloss].

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 [SAMLCore]. 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 1.1 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"?>
<schema targetNamespace="urn:liberty:iff:1.2" xmlns:lib="urn:liberty:iff:1.2"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ac="urn:liberty:ac:1.2"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
   <import namespace="urn:oasis:names:tc:SAML:1.0:assertion" schemaLocation="
    http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-
    01.xsd"/>
   <import namespace="urn:oasis:names:tc:SAML:1.0:protocol" schemaLocation="
    http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-protocol-
    01.xsd"/>
   <import namespace="http://www.w3.org/2001/04/xmlenc#"
    schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd"/>
   <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
   <import namespace="urn:liberty:ac:1.2"
    schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-
    authentication-context-v1.2.xsd"/>
   <include schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-
    utility-v1.0.xsd"/>
```

2.2 Type and Element Declarations

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

```
<simpleType name="ProviderIDType">
 <restriction base="anyURI"/>
</simpleType>
<element name="ProviderID" type="lib:ProviderIDType"/>
<element name="AffiliationID" type="lib:ProviderIDType"/>
```

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.

Introduction Notification: The protocol by which a first provider can notify a second provider about the successful introduction of a third provider to the first provider thanks to the mediation of the second provider.

Provider Relationship Termination: The protocol by which a first provider can notify a second provider than that a relationship between the first and a third provider has been terminated.

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 with many flexibilities and choices. All signed XML entities MUST adhere to the “XML Signature Profile” constraints defined in [SAMLCore].

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 the protocol messages and assertions. Version numbering of assertions is independent of the version numbering of the protocol messages.

This specification follows the version numbering requirements, processing rules, and error conditions specified in “SAML Versioning” in [SAMLCore].

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 [LibMetadata] 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. Name identifier values MUST NOT exceed a length of 256 characters.

When generating one-time-use identifiers for Principals, an identity provider MUST employ a pseudo-random technique such that the probability of two randomly chosen identifiers being identical MUST be less than 2^{-128} and SHOULD be less than 2^{-160}.

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. 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].

XML signatures SHOULD NOT be performed with any transforms other than:

- Enveloped Signature [XMLDsig]
- Exclusive XML Canonicalization [XMLCanon]

Receivers MUST NOT accept XML signatures created using other transforms without verification that the transforms do not omit any part of the data to be signed from the signed byte stream. Receivers MAY reject any messages with transforms other than the set specified above. Senders MUST NOT send messages using other transforms without prior agreement as to their contents.

XML signatures in messages MUST use a proper URI fragment in the URI attribute of the Reference element to identify the signed element. This URI fragment SHOULD reference the `id` attribute of an element in the same document using an XPointer [XPointer] shortcut reference.

The signer MUST NOT assume that the signed element will be at the root of the document during verification. It MUST be possible to validate the signature after adding or removing surrounding context for the profile in use (for example, the SOAP envelope, or the `<samlp:Response>` element). Implementers are encouraged to verify compliance with this requirement via empirical testing.

The **SignedSAMLRequestType** is provided to allow SAML requests to be signed using these guidelines. Specific usage of this type is shown in the relevant sections of [LibertyBindProf].

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 in to 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 [SAMLCore] regarding the use of nested second-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 implementors and deployers of the 1.2 specification in maximizing the opportunities for interoperability with software that implements an older specification. The primary goal is to avoid sending messages to communication endpoints that 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 [LibMetadata].
- 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.

3.1.12 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.

Three 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: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 works as follows:

1. A service provider issues an `<AuthnRequest>` request 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.

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 would generate a new authentication assertion without any intervention by the Principal.

3.2.1 Request

The service provider issues an `<AuthnRequest>` request to the identity provider. A set of parameters is included in the request that allows the service provider to specify desired behavior at the identity provider in processing the request. The service provider 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 service provider.

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 or limit the identity provider's ability 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 [SAMLCore], 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. Implementors 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 service provider's URI-based identifier.

`AffiliationId` [Optional]

If present, indicates that the requesting service provider is acting as a member of the affiliation group identified.

`NameIDPolicy` [Optional]

An enumeration permitting service provider influence over name identifier policy at the identity provider. If the element is omitted or "none", then a federated identity MUST already exist between the service provider and the identity provider. If set as "temporary", indicates that an anonymous, one-time identifier MUST be provided. If set to "federated", requests the identity be federated and a new persistent identifier generated if necessary. If the value is set as "any", this indicates that the provider favors a persistent identifier over a temporary one.

`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 service provider 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.

`AuthnContext` [Optional]

Information describing which authentication context the service provider 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.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification for more information.

`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>` or “better”, which means that the they can use any context better than any that were supplied. If not specified, this is assumed to be “exact”.

`ProxyCount` [Optional]

Specifies a limitation on the number of additional identity providers through which the authentication request may be proxied. May be zero, in which case no proxying is acceptable.

`IntroductionArtifact` [Optional]

Provides indirect evidence from a mutually trusted third party permitting the identity provider to consider trusting an unknown service provider or affiliation.

`consent` [Optional]

Indicates that consent has been obtained from a user in sending this message.

The `<AuthnContext>` element has the following mutually exclusive elements:

`AuthnContextClassRef` [Optional]

The ordered set of authentication context class references the service provider desires the identity provider to use in authenticating the Principal.

`AuthnContextStatementRef` [Optional]

The ordered set of exact authentication statements the service provider desires the identity provider to use in authenticating the Principal.

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

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

```
          <element ref="lib:AuthnContext" minOccurs="0"/>
          <element ref="lib:RelayState" minOccurs="0"/>
          <element name="AuthnContextComparison"
  type="lib:AuthnContextComparisonType" minOccurs="0"/>
          <element name="ProxyCount" type="nonNegativeInteger" minOccurs="0"/>
          <element ref="lib:IntroductionArtifact" minOccurs="0"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
        <attribute ref="lib:consent" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <simpleType name="NameIDPolicyType">
    <restriction base="string">
      <enumeration value="none"/>
      <enumeration value="temporary"/>
      <enumeration value="federated"/>
      <enumeration value="any"/>
    </restriction>
  </simpleType>
  <simpleType name="AuthnContextComparisonType">
    <restriction base="string">
      <enumeration value="exact"/>
      <enumeration value="minimum"/>
      <enumeration value="better"/>
    </restriction>
  </simpleType>
  <element name="NameIDPolicy type="lib:NameIDPolicyType"/>
  <element name="RelayState" type="string"/>
  <element name="ProtocolProfile" type="anyURI"/>
  <element name="AuthnContext">
    <complexType>
      <choice>
        <element name="AuthnContextClassRef" type="anyURI" maxOccurs="unbounded"/>
        <element name="AuthnContextStatementRef" type="anyURI"
  maxOccurs="unbounded"/>
      </choice>
    </complexType>
  </element>
  <element name="IntroductionArtifact" type="string"/>
```

3.2.1.2 Example

```
<lib:AuthnRequest id=”12345” RequestID="RPCUk2ll+GVz+t1lLURp51oFvJXk"
        MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
        IssueInstant="2001-12-17T21:42:4Z” xmlns:lib=”urn:liberty:iff:1.2”>
   <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:AuthnContext>

           <lib:AuthnContextClassRef>http://projectliberty.org/schemas/authctx/cla
        sses/Password-ProtectedTransport</lib:AuthnContextClassRef>
   </lib:AuthnContext>
   <lib:RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</lib:RelayState>
   <lib:AuthnContextComparison>exact</lib:AuthnContextComparison>
   <lib:ProxyCount>1</lib:ProxyCount>
</lib:AuthnRequest>
```

3.2.2 Response

The response is an <AuthnResponse> element containing either 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 <saml:Subject> element in any subject statement MUST be of type **SubjectType**.

If the <NameIDPolicy> element is omitted or "none", and 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 be "urn:liberty:iff:nameid:federated".

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 "temporary", 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:anonymous".

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 "temporary".

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

Identity providers MUST include a <saml:AudienceRestrictionCondition> element that specifies the intended consumers of the assertion. The <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.

Identity providers MAY include a SessionIndex attribute in resulting authentication statements, which is used to aid the identity provider in managing multiple sessions with the Principal. If the identity provider includes this SessionIndex attribute, 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.

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 URI-based identifier.

`RelayState` [Optional]

This contains state information being relayed.

`id` [Optional]

Identifier used to identify this element in a signature. See section 3.1.5, Signature Verification for more information.

`consent` [Optional]

Indicates that consent has been obtained from a user in sending this message.

The schema fragment is as follows:

```
<element name="AuthnResponse" type="lib:AuthnResponseType"/>
<complexType name="AuthnResponseType">
 <complexContent>
  <extension base="samlp:ResponseType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <element ref="lib:RelayState" minOccurs="0"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
     <attribute ref="lib:consent" use="optional"/>
  </extension>
 </complexContent>
</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>` is 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. The id attribute is also included to facilitate such signatures (see section 3.1.5, Signature Verification).

The schema fragment is as follows:

```
<complexType name="AssertionType">
 <complexContent>
  <extension base="saml:AssertionType">
     <attribute name="InResponseTo" type="saml:IDReferenceType"/>
     <attribute name="id" type="ID" use="optional"/>
  </extension>
 </complexContent>
</complexType>
```

3.2.2.3 Type SubjectType

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

Two `Format` URIs are defined for use with Liberty name identifiers:

- Identifiers communicated on behalf of Principals that have federated their identity MUST contain a `Format` URI of `urn:liberty:iff:nameid:federated`.

- Identifiers with anonymous, single-use semantics communicated on behalf of Principals that have not federated or wish to act anonymously MUST contain a `Format` URI of `urn:liberty:iff:nameid:anonymous`.

The schema fragment is as follows:

```
<complexType name="SubjectType">
 <complexContent>
  <extension base="saml:SubjectType">
     <sequence>
        <element ref="lib:IDPProvidedNameIdentifier"/>
     </sequence>
  </extension>
 </complexContent>
</complexType>
```

3.2.2.4 Type EncryptedSubjectType

The type **EncryptedSubjectType**, extended from **saml:SubjectType**, is used to encrypt the `<saml:NameIdentifier>` element in subject statements or other messages, if the name identifier is intended for use by a provider other than the one receiving the message. In such a case, the `<saml:NameIdentifier>` is ignored (any value MAY be present), and the actual name identifier can be recovered by decrypting the `<lib:EncryptedNameIdentifier>` element. The decryption key MAY itself be included (in encrypted form).

`EncryptedNameIdentifier` [Required]

An encrypted `<saml:NameIdentifier>` element and associated content.

`xenc:EncryptedKey` [Optional]

An optional decryption key, itself encrypted with a second key.

The schema fragment is as follows:

```
  <complexType name="EncryptedSubjectType">
      <complexContent>
          <extension base="saml:SubjectType">
              <sequence>
                  <element name="EncryptedNameIdentifier"
  type="xenc:EncryptedDataType"/>
                  <element ref="xenc:EncryptedKey" minOccurs="0"/>
              </sequence>
          </extension>
      </complexContent>
  </complexType>
```

3.2.2.5 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 Section 3.2 above).

`SessionIndex` [Optional]

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 “http://projectliberty.org/schemas/authctx/2002/05".

When the Service Provider is processing a `<saml:AuthenticationStatement>` of type **lib:AuthenticationStatementType** and the `saml:AuthenticationMethod` attribute is "http://projectliberty.org/schemas/authctx/2002/05", the Service Provider MUST refer to the `<AuthnContext>` element and ignore the `saml:AuthenticationMethod` attribute.

The schema fragment is as follows:

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

3.2.2.6 Example

```
<lib:AuthnResponse id=”54321” ResponseID="9hhuuja1bc744hGJn5Q9A5yvEIgS"
        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>
   <saml:Assertion id=”12345” MajorVersion="1" MinorVersion="0"
        AssertionID="e06e5a28-bc80-4ba6-9ecb-712949db686e"
        Issuer="http://IdentityProvider.com" IssueInstant="2001-12-17T09:30:47Z"
        InResponseTo="4e7c3772-4fa4-4a0f-99e8-7d719ff6067c"
        xsi:type="AssertionType">
      <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>
      <saml:AuthenticationStatement AuthenticationInstant="2001-12-17T09:30:47Z"
        SessionIndex="3" ReauthenticateOnOrAfter="2001-12-17T11:30:47Z"
        xsi:type="AuthenticationStatementType"
        AuthenticationMethod=”urn:oasis:names:tc:SAML:1.0:am:password”>
        <saml:Subject xsi:type="SubjectType">
           <saml:NameIdentifier
        Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be35-
        cc9e7db39e2b</saml:NameIdentifier>
```

```
            <IDPProvidedNameIdentifier
          Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be35-
          cc9e7db39e2b</IDPProvidedNameIdentifier>
          </saml:Subject>
       </saml:AuthenticationStatement>
       <ds:Signature>...</ds:Signature>
    </saml:Assertion>
    <lib:ProviderID>http://IdentityProvider.com</lib:ProviderID>
    <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</AuthnResponse>
```

3.2.3 Processing Rules

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 Provider ID of a known provider with which the identity provider has established a relationship, or an `<IntroductionArtifact>` element MAY be provided to allow the identity provider to obtain evidence to establish such a relationship, possibly only temporarily.

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 that contain any statement of a type other than **lib:IntroductionStatementType** MUST NOT appear in the `<AuthnResponse>`.

`<AffiliationId>`, if present, MUST be the Affiliation ID of a known affiliation 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 [LibMetadata]) and present a list of members of `<AffiliationId>` 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 "temporary", 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:anonymous".

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 "temporary" MUST be followed, in that order. Thus, a new federation may be created, an existing federation used, or a temporary identifier generated.

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 requesting provider-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 <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.

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

If one or more <AuthnContextClassRef> elements are included, then the resulting authentication statement in the assertion (if any) MUST contain an authentication statement that conforms to one of the specified classes. Additionally, the set of <AuthnContextClassRef> elements MUST be evaluated as an ordered set, where the first element is the most preferred authentication context class. If none of the specified authentication context classes can be satisfied, the identity provider MUST not include an authentication statement in the resulting assertion.

If one or more <AuthnContextStatementRef> 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 the identity provider wishes to rely on a second identity provider as the source of the Principal's authentication, then the following request processing rules apply. See also section 3.2.3.2 for additional detail.

If the `<ProxyCount>` element has a value of zero, then the identity provider MUST NOT attempt to proxy authenticate the Principal and MUST return an error with a second level `<samlp:StatusCode>` `Value` of "lib:ProxyCountExceeded", unless it can directly authenticate the principal.

If the `<ProxyCount>` element is omitted or has a value greater than zero, then the identity provider MAY refer the Principal to a second identity provider for authentication. If a `<ProxyCount>` was provided by the requester, then the identity provider MUST decrement the value by one and include the new value in a `<ProxyCount>` in the request it creates. Otherwise, the identity provider SHOULD provide a `<ProxyCount>` value of its own.

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

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.

3.2.3.1 Introduction of Identity Providers

An identity provider that is asked to authenticate an unknown Principal that it knows has an identity at another identity provider MAY introduce the requesting provider to the second identity provider. The process of introduction provides a basis for trust between the requesting provider and an identity provider with which it does not have an existing relationship. Introduction is accomplished by issuing (or obtaining from an authority) a set of "Introduction Assertions" to the requesting provider for its own use and for presentation to the second identity provider. Such an assertion contains statements of type **IntroductionStatementType** and MAY contain constraints on the introduction expressed as SAML conditions.

3.2.3.1.1 Type IntroductionStatementType

The type **IntroductionStatementType** is an extension of **saml:SubjectStatementType** that allows for the following elements:

`ProviderID` or `AffiliationID` [One or more]

The set of Liberty service and/or identity providers or affiliation groups being introduced to the statement subject by the issuer of the assertion.

`Notification` [Optional]

Indicates that the issuer of the assertion containing the statement wishes to be notified if a federation event occurs as a result of the introduction. The federating identity provider should notify the issuer using the protocol described in section 3.6.

The schema fragment is as follows:

```
<complexType name="IntroductionStatementType">
 <complexContent>
  <extension base="saml:SubjectStatementType">
     <choice maxOccurs="unbounded">
        <element ref="lib:ProviderID"/>
        <element ref="lib:AffiliationID"/>
     </choice>
     <element name="Notification" type="boolean" minOccurs="0"/>
```

```
      </extension>
    </complexContent>
  </complexType>
```

3.2.3.1.2 Processing Rules

To introduce a provider or an affiliation group to another identity provider the following steps are followed:

The identity provider MUST respond to the initial authentication request with a second level `<samlp:StatusCode> Value` of "lib:UnknownPrincipal", and MUST include one or more signed `<Assertion>` elements containing one or more statements of type **lib:IntroductionStatementType**.

The `<saml:Subject>` of the statement contains a `<saml:NameIdentifier>` corresponding to the `<ProviderID>` of the provider being introduced, with the provider or affiliation group it is being introduced to placed inside the statement's `<ProviderID>` or `<AffiliationID>` element.

Two statements MUST be included (in one assertion or in two different assertions), one with the service provider as the `<saml:Subject>` and the identity provider in the statement body, and one with the identity provider as the `<saml:Subject>` and the service provider or affiliation group in the statement body. If the original request contained an `<AffiliationID>` element, then the second introduction statement MUST include the `<AffiliationID>` in the body.

Upon receiving such an assertion, the provider MAY search for a statement with itself as the subject. The provider MUST examine any conditions on its use and decide whether to proceed with a new authentication request directed to the identity provider in the statement body.

If it proceeds, the provider MUST locate the assertion containing a statement with the new identity provider as the subject and itself or its affiliation group in the body, and it MUST generate a SAML artifact for the assertion using its own `<ProviderID>` when constructing the artifact. See section 3.2.2.2 of [LibertyBindProf] for a description of the artifact format and syntax rules.

Upon receiving such a request, the second identity provider MUST obtain and examine the assertion and any conditions on its use before honoring the request. Obtaining the assertion involves dereferencing the artifact, as defined by [SAMLBind]. The provider that constructs the artifact MUST have a SOAP endpoint available at which the identity provider can request the assertion.

If the identity provider federates the Principal's identity as a result of an introduction statement that contains a `<Notification>` element with a value of "true", then it MUST use the protocol described in section 3.6 to notify the introducing identity provider of the federation event.

3.2.3.2 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, provided the value in the `<ProxyCount>` element is greater than zero, if one exists. The limit of such proxying steps is governed only by the `<ProxyCount>` on the request and local policy. When creating the new authentication request:

The identity provider MUST include equivalent or more strict forms of all the information included in the requesting provider's request (such as authentication context policy).

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

If the request contained a `<ProxyCount>` element with a value greater than 0, then the new request MUST contain a `<ProxyCount>` element with a value of 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.

The authentication request and response are processed in normal fashion, in accordance with the rules given in section 3.2.3. 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 the identity provider, while the new assertion's condition will reference the original requesting provider.

If the `<NameIdentifier>` is anonymous (determined by examining the `Format` URI), then the identity provider MUST generate a new anonymous identifier and include it in the new assertion.

If the `<NameIdentifier>` is not anonymous, 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.

The new assertion MUST contain `<AuthnContext>` information with, at minimum, an `<ac:AuthenticatingIDP>` element identifying the authenticating identity provider and any `<ac:GoverningAgreements>` data that may be applicable.

If the original assertion contains `<AuthnContext>` information that includes one or more `<ac:AuthenticatingIDP>` 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.

3.2.3.3 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 [SAMLCore], utilizing second-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 3.2.4.

Specify whether the `<AuthnResponse>` element sent from the identity provider to the service provider via the intermediary is wrapped in an `<AuthnResponseEnvelope>`. See section 3.2.5.

3.2.3.3.1 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 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.

3.2.3.4 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-level <samlp:StatusCode> elements, if the responder wishes to provide additional detail. If reporting a specific status value will not expose the responder or the Principal to security risk or exposure of unneeded 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. MAY be accompanied by an introduction assertion.

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.

3.2.4 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 3.2.3.1.1. 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 insure that the XML obtained by removing the `<AuthnRequestEnvelope>` from the enclosed `<AuthnRequest>` is well-formed and valid.

3.2.4.1 Element <AuthnRequestEnvelope>

The authentication request envelope contains the following elements:

`Extension` [Optional]

Optional container for protocol extensions established by agreement between providers. Implementors 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 requestor's ProviderID.

`ProviderName` [Optional]

The human-readable name of the requestor.

`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:

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

```
</complexType>
```

3.2.4.2 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`

The container element for an IDP List.

`IDPEntries`

Contains a list of identity provider entries.

`IDPEntry`

Describes an identity provider that the service provider supports.

`ProviderID`

The identity provider's ProviderID.

`ProviderName`

The identity provider's human-readable name.

`Loc`

The identity provider's URI, to which authentication requests may be sent. This SHOULD be set to the value of the identity provider's `<SingleSignOnService>` metadata element.

`GetComplete`

If the identity provider list is not complete, this element is included with a URI that points to where the complete list can be retrieved.

The schema fragment is as follows:

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

3.2.4.3 Example

```
<AuthnRequestEnvelope>
   <AuthnRequest> ... </AuthnRequest>
   <ProviderID>http://ServiceProvider.com</ProviderID>
   <ProviderName>Service Provider X</ProviderName>
           <AssertionConsumerServiceURL>http://ServiceProvider.com/lecp_assertion_
        consumer</AssertionConsumerServiceURL>
   <IDPList>
     <IDPEntries>
      <IDPEntry>
         <ProviderID>http://IdentityProvider.com</ProviderID>
         <ProviderName>Identity Provider X</ProviderName>
         <Loc>http://www.IdentityProvider.com/liberty/sso</Loc>
      </IDPEntry>
     </IDPEntries>
     <GetComplete>https://ServiceProvider.com/idplist?id=604be136-fe91-441e-afb8-
        f88748ae3b8b </GetComplete>
   </IDPList>
   <IsPassive>0</IsPassive>
</AuthnRequestEnvelope>
```

3.2.5 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 3.2.3.1.1. 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.

3.2.5.1 Element <AuthnResponseEnvelope>

The authentication response envelope contains the following elements:

`Extension` [Optional]

Optional container for protocol extensions established by agreement between providers. Implementors 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 Provider Metadata.

The schema fragment is as follows:

```
<element name="AuthnResponseEnvelope" type="lib:AuthnResponseEnvelopeType"/>
<complexType name="AuthnResponseEnvelopeType">
 <complexContent>
  <extension base="lib:ResponseEnvelopeType">
   <sequence>
    <element ref="lib:AuthnResponse"/>
    <element name="AssertionConsumerServiceURL" type="anyURI"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="ResponseEnvelopeType">
 <sequence>
```

```
      <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
```

3.2.5.2 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 handle 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 handle with the identity provider. This opaque handle 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.

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]

The name identifier the service provider should use when communicating with the identity provider.

`SPProvidedNameIdentifier` [Required]

The name identifier the identity provider should use when communicating to the service provider.

`OldProvidedNameIdentifier` [Required]

In the case of either provider choosing to request a change of provided name identifiers, this element holds the previous version. For a service provider making their first name change following federation, the <`OldProvidedNameIdentifier`> will contain the current `<IDPProvidedNameIdentifier>`. The `<SPProvidedNameIdentifier>` will contain the new name that the service provider wishes the identity provider to use.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`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.

`AffiliationId` [Optional]

If present and the requester is a service provider, indicates that the requesting service provider is acting as a member of the specified affiliation. If present and the requester is an identity provider, indicates that the receiving service provider should interpret the message as being from a member of the specified affiliation, and representing the group, rather than the individual provider.

The schema fragment is as follows:

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

3.3.1.2 Example

```
<RegisterNameIdentifierRequest id=”12345” 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>342ad3d8-93ee-4c68-be35-
        cc9e7db39e2b</IDPProvidedNameIdentifier>
   <SPProvidedNameIdentifier>e958019a</SPProvidedNameIdentifier>
          <OldProvidedNameIdentifier>e895014a</OldProvidedNameIdentifier>
   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</RegisterNameIdentifierRequest>
```

3.3.2 Response

The responding provider MUST respond with `<RegisterNameIdentifierResponse>`, 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.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`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:

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

3.3.2.2 Example

```
<RegisterNameIdentifierResponse id="12345" ResponseID="74ffec0f-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://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 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.

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.

`<AffiliationId>`, if present, MUST be the Affiliation ID of a known affiliation with which the identity provider has an established relationship and of which the service provider is a member. If present, providers MUST refer to the federation associated with the specified affiliation, if any, not the service provider, in processing the message.

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.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`AffiliationId` [Optional]

If present and the requester is a service provider, indicates that the requesting service provider is acting as a member of the specified Affiliation. If present and the requester is an identity provider, indicates that the receiving service provider should interpret the message as a member of the specified affiliation.

`consent` [Optional]

Indicates that consent has been obtained from a user in sending this message.

The schema fragment is as follows:

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

3.4.1.2 Example

```
<FederationTerminationNotification id=”12345” RequestID="9ec2-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>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.

`<AffiliationId>`, if present, MUST be the Affiliation ID of a known affiliation with which the identity provider has an established relationship and of which the service provider is a member. If present, providers MUST refer to the federation associated with the specified affiliation, if any, not the service provider, in processing the message.

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.

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 for the session.

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 in the current session with the Principal, with the exception of the service provider that sent the `<LogoutRequest>` message to the Identity Provider.

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 the authentication statement in the assertion that the service provider used in establishing the session with the Principal contained a `SessionIndex` attribute. This message SHOULD be signed.

3.5.1.1 Element <LogoutRequest>

`Extension` [Optional]

Optional container for protocol extensions established by agreement between providers. Implementors 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>`.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`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.

`AffiliationId` [Optional]

If present and the requester is a service provider, indicates that the requesting service provider is acting as a member of the specified Affiliation. If present and the requester is an identity provider, indicates that the receiving service provider should interpret the message as a member of the specified affiliation.

`consent` [Optional]

Indicates that consent has been obtained from a user in sending this message.

The schema fragment is as follows:

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

3.5.1.2 Example

```
<LogoutRequest id=”12345” RequestID="47693d03-7c33-4d65-931f-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>342ad3d8-93ee-4c68-be35-cc9e7db39e2b</saml:NameIdentifier>
   <SessionIndex>3</SessionIndex>
   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</LogoutRequest>
```

3.5.1.3 Response

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

This message SHOULD be signed.

3.5.1.4 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.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`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:

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

3.5.1.5 Example

```
<LogoutResponse id=”12345” ResponseID="74ffec0f-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>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</LogoutResponse>
```

3.5.1.6 Processing Rules

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

<AffiliationId>, if present, MUST be the Affiliation ID of a known affiliation with which the identity provider has an established relationship and of which the service provider is a member. If present, providers MUST interpret the <saml:NameIdentifier> in the context of the specified affiliation, not the service provider, in processing the message.

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

3.5.1.7 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.
- 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 requestor 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.

3.5.1.8 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 referred to in the <saml:NameIdentifier> element.

3.6 Introduction Notification Protocol

If a service provider is successfully introduced to an identity provider through the mediation of an *introducing* identity provider; the service provider federates a Principal's identity with the provider to whom they have been introduced, and if the introducing provider has requested notification of federation by including a <Notification> element of "true" in the <IntroductionStatement>, then the identity provider to whom the service provider has been introduced MUST send an <IntroductionNotification> message to the introducing provider.

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

3.6.1 Message

An identity provider that is federating a Principal's identity as a result of the introduction of another identity provider sends a `<IntroductionNotification>` message to the introducing provider if that provider has requested introduction notification via the `<Notification>` element.

The `<IntroductionNotification>` message SHOULD be signed.

3.6.1.1 Element <IntroductionNotification>

The elements are as follows:

`Extension` [Optional]

Optional container for protocol extensions established by agreement between providers.

`ProviderID` [Required]

The identifier of the identity provider that is sending this message.

`IntroducedProviderID` or `IntroducedAffiliationID` [Required]

The identifier of the service provider or affiliation group that has been successfully introduced.

`AssertionIDReference` [Required]

The identifier of the introduction assertion utilized by the sending provider to accept the introduction of the provider to whom the notification is being sent.

`NameIdentifier` [Required]

The name identifier of the Principal whose identity federation triggered the introduction. This name identifier MUST be unique per Principal, generated specifically for use by this protocol and MUST be different for each introducing identity provider. It MUST be reused if the same Principal triggers introduction of subsequent service providers.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

The schema fragment is as follows:

```
<element name="IntroductionNotification" type="lib:IntroductionNotificationType"/>
<complexType name="IntroductionNotificationType">
 <complexContent>
  <extension base="samlp:RequestAbstractType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <choice>
           <element name="IntroducedProviderID" type="lib:ProviderIDType"/>
           <element name="IntroducedAffiliationID" type="lib:ProviderIDType"/>
        </choice>
        <element ref="saml:AssertionIDReference"/>
        <element ref="saml:NameIdentifier"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
  </extension>
 </complexContent>
</complexType>
```

3.6.1.2 Example

```
<IntroductionNotification id=”12345” RequestID="9ec2-eb65-4bce-ab8f-4becdf229815"
        MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">
   <ds:Signature>...</ds:Signature>
   <ProviderID>http://IdentityProvider.com</ProviderID>
   <IntroducedProviderID>http://ServiceProvider.com</IntroducedProviderID>
   <saml:AssertionIDReference>dswd398t4deurd4</saml:AssertionIDReference>
   <saml:NameIdentifier>e958019a</saml:NameIdentifier>
</IntroductionNotification>
```

3.6.2 Processing Rules

The receiving identity 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.

3.7 Provider Relationship Termination Protocol

When a service provider has previously been successfully introduced to an identity provider through the mediation of an introducing identity provider, but the business relationship between the identity providers has subsequently been broken, then the introducing identity provider SHOULD send a <`ProviderRelationshipTerminationRequest`> message to the service provider.

Upon reception of a <`ProviderRelationshipTerminationRequest`> message, a provider MUST respond with a <`ProviderRelationshipTerminationResponse`> message.

3.7.1 Message

The introducing identity provider sends a `<ProviderRelationshipTerminationRequest>` to the service provider concerning the termination of the relationship between the identity providers.

The `<ProviderRelationshipTerminationRequest>` message MUST be signed.

3.7.1.1 Element <ProviderRelationshipTerminationRequest>

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.

`TerminatedProviderID` [Required]

The identifier of the provider with which the sender has terminated a relationship.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`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:

```
<element name="ProviderRelationshipTerminationRequest"
 type="lib:ProviderRelationshipTerminationRequestType"/>
<complexType name="ProviderRelationshipTerminationRequestType">
 <complexContent>
  <extension base="samlp:RequestAbstractType">
    <sequence>
      <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="lib:ProviderID"/>
      <element name="TerminatedProviderID" type="lib:ProviderIDType"/>
      <element ref="lib:RelayState" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID" use="optional"/>
  </extension>
 </complexContent>
</complexType>
```

3.7.1.2 Example

```
<ProviderRelationshipTerminationRequest id="abcde" RequestID="9ec2-eb65-4bce-ab8f-
        4becdf229815" MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-
        17T09:30:47Z">
  <ds:Signature>...</ds:Signature>
  <ProviderID>http://IntroducingIdentityProvider.com</ProviderID>
  <TerminatedProviderID>http://IntroducedIdentityProvider.com</TerminatedProvider
        ID>
  <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</ProviderRelationshipTerminationRequest>
```

3.7.2 Response

The responding provider MUST return a `<ProviderRelationshipTerminationResponse>` message, which is of type **StatusResponseType**.

This message SHOULD be signed.

3.7.2.1 Element <ProviderRelationshipTerminationResponse>

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 of the notification message.

`id` [Optional]

Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

`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:

```
<element name="ProviderRelationshipTerminationResponse"
 type="lib:StatusResponseType"/>
```

3.7.2.2 Example

```
<ProviderRelationshipTerminationResponse id=”abcde” ResponseID="74ffec0f-1165-
        4fa3-b088-3dd2c2388b91" InResponseTo="eb20e77f-d982-44f9-936e-
        dd135bf437d4" MajorVersion="1" MinorVersion="0" IssueInstant="2001-12-
        17T09:30:47Z" Recipient="http://IntroducingIdentityProvider.com">
   <ds:Signature>...</ds:Signature>
   <ProviderID>http://ServiceProvider.com</ProviderID>
   <samlp:Status>
      <samlp:StatusCode Value="samlp:Success"/>
   </samlp:Status>
   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
</ProviderRelationshipTerminationResponse>
```

3.7.3 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.

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

3.7.3.1 Recipient Processing Rules

The receiving provider MAY choose to invalidate the relationship with the provider identified by the <`TerminatedProviderID`> element.

The recipient MUST respond with a <`ProviderRelationshipTerminationResponse`> message.

4 Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:liberty:iff:1.2" xmlns:lib="urn:liberty:iff:1.2"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ac="urn:liberty:ac:1.2"
    xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
   <import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
    schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-
    schema-assertion-01.xsd"/>
   <import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
    schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-
    schema-protocol-01.xsd"/>
   <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
   <import namespace="urn:liberty:ac:1.2"
    schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-
    authentication-context-v1.2.xsd"/>
   <include schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-
    utility-v1.0.xsd"/>
   <simpleType name="ProviderIDType">
    <restriction base="anyURI"/>
   </simpleType>
   <element name="ProviderID" type="lib:ProviderIDType"/>
   <element name="AffiliationID" type="lib:ProviderIDType"/>
   <complexType name="SignedSAMLRequestType">
    <complexContent>
     <extension base="samlp:RequestType">
        <attribute name="id" type="ID" use="optional"/>
     </extension>
```

```
    </complexContent>
  </complexType>
  <element name="AuthnRequest" type="lib:AuthnRequestType"/>
  <complexType name="AuthnRequestType">
    <complexContent>
      <extension base="samlp:RequestAbstractType">
        <sequence>
          <element ref="lib:Extension minOccurs="0" maxOccurs="unbounded"/>
          <element ref="lib:ProviderID"/>
          <element ref="lib:AffiliationId" minOccurs="0"/>
          <element ref="lib:NameIDPolicy" minOccurs="0"/>
          <element name="ForceAuthn" type="boolean" minOccurs="0"/>
          <element name="IsPassive" type="boolean" minOccurs="0"/>
          <element ref="lib:ProtocolProfile" minOccurs="0"/>
          <element name="AssertionConsumerServiceID" type="string" minOccurs="0"/>
          <element ref="lib:AuthnContext" minOccurs="0"/>
          <element ref="lib:RelayState" minOccurs="0"/>
          <element name="AuthnContextComparison"
    type="lib:AuthnContextComparisonType" minOccurs="0"/>
          <element name="ProxyCount" type="nonNegativeInteger" minOccurs="0"/>
          <element ref="lib:IntroductionArtifact" minOccurs="0"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
        <attribute ref="lib:consent" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <simpleType name="NameIDPolicyType">
    <restriction base="string">
      <enumeration value="none"/>
      <enumeration value="temporary"/>
      <enumeration value="federated"/>
      <enumeration value="any"/>
    </restriction>
  </simpleType>
  <simpleType name="AuthnContextComparisonType">
    <restriction base="string">
      <enumeration value="exact"/>
      <enumeration value="minimum"/>
      <enumeration value="better"/>
    </restriction>
  </simpleType>
  <element name="NameIDPolicy type="lib:NameIDPolicyType"/>
  <element name="RelayState" type="string"/>
  <element name="ProtocolProfile" type="anyURI"/>
  <element name="AuthnContext">
    <complexType>
      <choice>
        <element name="AuthnContextClassRef" type="anyURI" maxOccurs="unbounded"/>
        <element name="AuthnContextStatementRef" type="anyURI"
    maxOccurs="unbounded"/>
      </choice>
    </complexType>
  </element>
  <element name="IntroductionArtifact" type="string"/>
  <element name="AuthnResponse" type="lib:AuthnResponseType"/>
  <complexType name="AuthnResponseType">
    <complexContent>
      <extension base="samlp:ResponseType">
        <sequence>
          <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
          <element ref="lib:ProviderID"/>
          <element ref="lib:RelayState" minOccurs="0"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
        <attribute ref="lib:consent" use="optional"/>
      </extension>
```

```
    </complexContent>
  </complexType>
  <element name="Assertion" type="lib:AssertionType"/>
  <complexType name="AssertionType">
    <complexContent>
      <extension base="saml:AssertionType">
        <attribute name="InResponseTo" type="saml:IDReferenceType"/>
        <attribute name="id" type="ID" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="SubjectType">
    <complexContent>
      <extension base="saml:SubjectType">
        <sequence>
          <element ref="lib:IDPProvidedNameIdentifier"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="AuthenticationStatementType">
    <complexContent>
      <extension base="saml:AuthenticationStatementType">
        <sequence>
          <element name="AuthnContext" minOccurs="0">
            <complexType>
              <sequence>
                <element name="AuthnContextClassRef" type="anyURI" minOccurs="0"/>
                <choice>
                  <element ref="ac:AuthenticationContextStatement"/>
                  <element name="AuthnContextStatementRef" type="anyURI"/>
                </choice>
              </sequence>
            </complexType>
          </element>
        </sequence>
        <attribute name="ReauthenticateOnOrAfter" type="dateTime" use="optional"/>
        <attribute name="SessionIndex" type="string" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="IntroductionStatementType">
    <complexContent>
      <extension base="saml:SubjectStatementType">
        <choice maxOccurs="unbounded">
          <element ref="lib:ProviderID"/>
          <element ref="lib:AffiliationID"/>
        </choice>
        <element name="Notification" type="boolean" minOccurs="0"/>
      </extension>
    </complexContent>
  </complexType>
  <element name="AuthnRequestEnvelope" type="lib:AuthnRequestEnvelopeType"/>
  <complexType name="AuthnRequestEnvelopeType">
    <complexContent>
      <extension base="lib:RequestEnvelopeType">
        <sequence>
          <element ref="lib:AuthnRequest"/>
          <element ref="lib:ProviderID"/>
          <element name="ProviderName" type="string" minOccurs="0"/>
          <element name="AssertionConsumerServiceURL" type="anyURI"/>
          <element ref="lib:IDPList" minOccurs="0"/>
          <element name="IsPassive" type="boolean" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

```
<complexType name="RequestEnvelopeType">
 <sequence>
  <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
<element name="IDPList" type="lib:IDPListType"/>
<complexType name="IDPListType">
 <sequence>
  <element ref="lib:IDPEntries"/>
  <element ref="lib:GetComplete" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
<element name="IDPEntry">
 <complexType>
  <sequence>
     <element ref="lib:ProviderID"/>
     <element name="ProviderName" type="string" minOccurs="0"/>
     <element name="Loc" type="anyURI"/>
  </sequence>
 </complexType>
</element>
<element name="IDPEntries">
 <complexType>
  <sequence>
     <element ref="lib:IDPEntry" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
<element name="GetComplete" type="anyURI"/>
<element name="AuthnResponseEnvelope" type="lib:AuthnResponseEnvelopeType"/>
<complexType name="AuthnResponseEnvelopeType">
 <complexContent>
  <extension base="lib:ResponseEnvelopeType">
     <sequence>
        <element ref="lib:AuthnResponse"/>
        <element name="AssertionConsumerServiceURL" type="anyURI"/>
     </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="ResponseEnvelopeType">
 <sequence>
  <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
<element name="RegisterNameIdentifierRequest"
 type="lib:RegisterNameIdentifierRequestType"/>
<complexType name="RegisterNameIdentifierRequestType">
 <complexContent>
  <extension base="samlp:RequestAbstractType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
        <element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
        <element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
        <element ref="lib:RelayState" minOccurs="0"/>
                  <element ref="lib:AffiliationId" minOccurs="0"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
  </extension>
 </complexContent>
</complexType>
<element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
<element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
<element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
<element name="RegisterNameIdentifierResponse" type="lib:StatusResponseType"/>
```

```
<complexType name="StatusResponseType">
 <complexContent>
  <extension base="samlp:ResponseAbstractType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <element ref="samlp:Status"/>
        <element ref="lib:RelayState" minOccurs="0"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
  </extension>
 </complexContent>
</complexType>
<element name="FederationTerminationNotification"
 type="lib:FederationTerminationNotificationType"/>
<complexType name="FederationTerminationNotificationType">
 <complexContent>
  <extension base="samlp:RequestAbstractType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <element ref="saml:NameIdentifier"/>
        <element ref="lib:AffiliationId" minOccurs="0"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
     <attribute ref="lib:consent" use="optional"/>
  </extension>
 </complexContent>
</complexType>
<element name="LogoutRequest" type="lib:LogoutRequestType"/>
<complexType name="LogoutRequestType">
 <complexContent>
  <extension base="samlp:RequestAbstractType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <element ref="saml:NameIdentifier"/>
        <element name="SessionIndex" type="string" minOccurs="0"/>
        <element ref="lib:RelayState" minOccurs="0"/>
        <element ref="lib:AffiliationId" minOccurs="0"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
     <attribute ref="lib:consent" use="optional"/>
  </extension>
 </complexContent>
</complexType>
<element name="LogoutResponse" type="lib:StatusResponseType"/>
<element name="IntroductionNotification" type="lib:IntroductionNotificationType"/>
<complexType name="IntroductionNotificationType">
 <complexContent>
  <extension base="samlp:RequestAbstractType">
     <sequence>
        <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="lib:ProviderID"/>
        <choice>
           <element name="IntroducedProviderID" type="lib:ProviderIDType"/>
           <element name="IntroducedAffiliationID" type="lib:ProviderIDType"/>
        </choice>
        <element ref="saml:AssertionIDReference"/>
        <element ref="saml:NameIdentifier"/>
     </sequence>
     <attribute name="id" type="ID" use="optional"/>
  </extension>
 </complexContent>
</complexType>
<element name="ProviderRelationshipTerminationRequest"
 type="lib:ProviderRelationshipTerminationRequestType"/>
```

```
   <complexType name="ProviderRelationshipTerminationRequestType">
    <complexContent>
     <extension base="samlp:RequestAbstractType">
        <sequence>
           <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
           <element ref="lib:ProviderID"/>
           <element name="TerminatedProviderID" type="lib:ProviderIDType"/>
           <element ref="lib:RelayState" minOccurs="0"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
     </extension>
    </complexContent>
   </complexType>
   <element name="ProviderRelationshipTerminationResponse"
    type="lib:StatusResponseType"/>
</schema>
```

5 References

[LibertyBindProf] Rouault, J., & Wason, T., eds. (January 2003). "Liberty Bindings and Profiles Specification," Version 1.2. Liberty Alliance Project, <http://www.projectliberty.org/specs/>.

[LibertyGloss] Mauldin, H., & Wason, T., eds. (January 2003). "Liberty Architecture Glossary," Version 1.1. Liberty Alliance Project, <http://www.projectliberty.org/specs/>.

[RFC2119] Bradner, S. (March 1997). "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc2119.txt> [18 December 2002].

[RFC3280] Housley, R. (April 2002). "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280. The Internet Engineering Task Force, <http://www.rfc-editor.org/rfc/rfc3280.txt> [18 December 2002]

[SAMLCore] Hallam-Baker, P., Maler, E., eds. (05 Nov. 2002). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)," Version 1.0, OASIS Standard. Organization for the Advancement of Structured Information Standards, <http://www.oasis-open.org/committees/security/#documents> [18 December 2002].

[SAMLBind] Mishra, P., ed.. (05 Nov. 2002). "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)," Version 1.0, OASIS Standard. Organization for the Advancement of Structured Information Standards, <http://www.oasis-open.org/committees/security/#documents> [18 December 2002].

[Schema1] Thompson, H. S., Beech, D., Maloney, M., & Mendleshon, N., eds. (May 2002). "XML Schema Part 1: Structures," Recommendation. World Wide Web Consortium, <http://www.w3.org/TR/xmlschema-1/> [18 December 2002].

[Schema2] Biron, P. V., & Malhotra, A., eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation. World Wide Web Consortium, <http://www.w3.org/TR/xmlschema-2/> [18 December 2002].

[XMLDsig] D. Eastlake, J. Reagle, D. Solo et al, (12 February 2002). "XML-Signature Syntax and Processing", Recommendation. World Wide Web Consortium, <http://www.w3.org/TR/xmldsig-core> [18 December 2002].

[XMLCanon] J. Boyer, D. Eastlake, & J. Reagle (18 July 2002). "Exclusive XML Canonicalization," Recommendation. World Wide Web Consortium, <http://www.w3.org/TR/xml-exc-c14n> [18 December 2002].

[XPointer] DeRose, S., Maler, E., & Daniel, R. (16 August 2002). "XML Pointer Language (XPointer)," Working Draft. World Wide Web Consortium, <http://www.w3.org/TR/xptr/> [18 December 2002].

[LibMetadata] Davis, P. (14 April 2003). "Liberty Core Metadata", Version 1.0-DRAFT. Liberty Alliance Project, <http://www.projectliberty.org/specs> [14 April 2003].