



Liberty ID-FF Protocols and Schema Specification

Version: 1.2

Editors:

Scott Cantor, OSU/Internet2
John Kemp, Liberty Alliance

Contributors:

Robert Aarts, Nokia
John Beatty, Sun Microsystems, Inc.
Conor Cahill, AOL Time Warner, Inc.
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 specifications.

Filename: liberty-idff-protocols-schema-v1.2.pdf

1 Notice

2 Copyright © 2003 America Online, Inc.; American Express Travel Related Services; Bank of America; Bell Canada;
3 Cingular Wireless; Cisco Systems, Inc.; Communicator, Inc.; Deloitte & Touche LLP; Earthlink, Inc.; Electronic
4 Data Systems, Inc.; Entrust, Inc.; Ericsson; Fidelity Investments; France Telecom; Gemplus; General Motors;
5 Hewlett-Packard Company; i2 Technologies, Inc.; Intuit Inc.; MasterCard International; NEC Corporation; Netegrity;
6 NeuStar; Nextel Communications; Nippon Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.;
7 NTT DoCoMo, Inc.; OneName Corporation; Openwave Systems Inc.; PricewaterhouseCoopers LLP; Register.com;
8 Royal Mail; RSA Security Inc; Sabre Holdings Corporation; SAP AG; SchlumbergerSema; SK Telecom; Sony
9 Corporation; Sun Microsystems, Inc.; Symlabs, Inc.; Trustgenix; United Airlines; VeriSign, Inc.; Visa International;
10 Vodafone Group Plc; Wave Systems;. All rights reserved.

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

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

24 Liberty Alliance Project
25 Licensing Administrator
26 c/o IEEE-ISTO
27 445 Hoes Lane
28 Piscataway, NJ 08855-1331, USA
29 info@projectliberty.org

30 **Contents**

31	1. Introduction	4
32	2. Schema Declarations	6
33	3. Protocols	7
34	4. Schema Definition	41
35	References	46

36 1. Introduction

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

40 1.1. Notation

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

45 The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT,"
46 "RECOMMENDED," "MAY," and "OPTIONAL" in this specification are to be interpreted as described in [\[RFC2119\]](#):
47 "they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for
48 causing harm (e.g., limiting retransmissions)."

49 These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application
50 features and behavior that affect the interoperability and security of implementations. When these words are not
51 capitalized, they are meant in their natural-language sense.

52 Listings of schemas appear like this.

53 Listings of instance fragments appear like this.

54 The following namespaces are referred to in this document:

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

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

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

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

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

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

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

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

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

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

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

70 Definitions for Liberty-specific terms can be found in [\[LibertyGlossary\]](#).

71 1.2. Overview

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

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

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

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

86 2. Schema Declarations

87 This document specifies an XML schema for Liberty ID-FF. The schema header along with namespace, type, and
88 element declarations are in 1.1 and 2.2.

89 2.1. Schema Header and Namespace Declarations

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

```
91 <?xml version="1.0" encoding="UTF-8"?>
92 <xs:schema targetNamespace="urn:liberty:iff:2003-08"
93   xmlns="urn:liberty:iff:2003-08"
94   xmlns:xs="http://www.w3.org/2001/XMLSchema"
95   xmlns:ac="urn:liberty:ac:2003-08"
96   xmlns:md="urn:liberty:metadata:2003-08"
97   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
98   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
99   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
100   elementFormDefault="qualified" attributeFormDefault="unqualified">
101
102   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
103     schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
104   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
105     schemaLocation="oasis-sstc-saml-schema-protocol-1.1.xsd"/>
106   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
107     schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd"/>
108   <xs:import namespace="urn:liberty:ac:2003-08" schemaLocation="liberty-authentication-conte
109 xt-v1.2.xsd"/>
110   <xs:import namespace="urn:liberty:metadata:2003-08" schemaLocation="liberty-metadata-v1.0.xsd"/>
111
112   <xs:include schemaLocation="liberty-idff-utility-v1.0.xsd"/>
113
114   <xs:annotation>
115     <xs:documentation>
116       The source code in this XSD file was excerpted verbatim from:
117       Liberty ID-FF Protocols & Schema Specification
118       Version 1.2
119       12th November 2003
120       Copyright (c) 2003 Liberty Alliance participants, see
121       http://www.projectliberty.org/specs/idff_copyrights.html
122     </xs:documentation>
123   </xs:annotation>
124
125 </xs:schema>
```

130 2.1.1. Type and Element Declarations

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

```
132 <xs:element name="ProviderID" type="md:entityIDType"/>
133 <xs:element name="AffiliationID" type="md:entityIDType"/>
134
135
136
```

137 **3. Protocols**

138 The Liberty protocol suite consists of the following protocols:

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

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

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

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

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

147 **3.1. General Requirements**

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

149 **3.1.1. XML Signature**

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

152 **3.1.2. Protocol and Assertion Versioning**

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

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

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

164 In assigning version attributes, implementers should follow these rules:

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

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

170 3.1.3. Provider and Affiliation ID Uniqueness

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

174 All provider and affiliation identifiers **MUST** conform to the rules specified in [\[LibertyMetadata\]](#) regarding such
175 identifiers.

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

178 3.1.4. Name Identifier Construction

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

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

188 3.1.5. Signature Verification

189 Processing rules for the protocols defined in this document commonly specify digital signature verification. In
190 these cases, it is not sufficient to only verify the signature of the signed object. The processing rules defined in
191 [\[SAMLCORE11\]](#) apply to all signed Liberty protocol messages. Verification of the `<ds:Signature>` element **MUST**
192 be performed in accordance with the best practices for the certification path technology in use. For example, when
193 using X.509 v3 public key certificates it is strongly **RECOMMENDED** that certification path validation be performed
194 in accordance to the PKIX Profile as specified in [\[RFC3280\]](#).

195 3.1.6. Security

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

201 3.1.7. Time Values

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

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

208 **3.1.8. Time Synchronization**

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

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

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

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

217 • `<LogoutRequest>` is received

218 • The user's session times out via normal means

219 • The `ReauthenticateOnOrAfter` time on the `<AuthenticationStatement>` used to authenticate the Princi-
220 pal, if any, is reached

221 **3.1.9. Response Status Codes**

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

226 **3.1.10. Use of `<Extension>` in Protocols**

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

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

235 **3.1.11. Interoperation with previous Liberty Implementations**

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

240 • Metadata SHOULD be used to the greatest extent possible to identify the capabilities of a provider, so that the
241 proper messages can be sent. See [\[LibertyMetadata\]](#).

242 • Version 1.2 implementations SHOULD avoid sending 1.2 requests or notifications to providers that only implement
243 earlier versions of the specification.

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

246 • Version 1.2 implementations MUST respond to older requests with responses matching the version of the request.
247 These rules apply to all of the request/response protocols and asynchronous notifications defined in this specifica-
248 tion.

249 3.1.12. Use of the consent attribute

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

252 Three values are defined for this attribute:

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

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

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

258 3.2. Single Sign-On and Federation Protocol

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

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

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

268 **Note:**

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

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

273 3.2.1. Request

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

277 • Prompt the Principal for credentials if the Principal is not presently authenticated.

- 278 • Prompt the Principal for credentials, even if the Principal is presently authenticated.
- 279 • Federate the Principal's identity at the identity provider with the Principal's identity at the requester.
- 280 • Issue an anonymous and temporary identifier for the Principal to the service provider.
- 281 • Use a specific protocol profile in responding to the request.
- 282 • Use a specific authentication context (for example, smartcard-based authentication vs. username/password-based
283 authentication).
- 284 • Restrict the ability of the recipient to proxy the authentication request to additional identity providers.

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

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

289 3.2.1.1. Element <AuthnRequest>

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

293 The elements of the request are as follows:

294 `Extension` [Optional]

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

298 `ProviderID` [Required]

299 The requester's unique identifier.

300 `AffiliationID` [Optional]

301 If present, indicates that the requester is acting as a member of the affiliation group identified.

302 `NameIDPolicy` [Optional]

303 An enumeration permitting requester influence over name identifier policy at the identity provider.

304 `IsPassive` [Optional]

305 If "true," specifies that the identity provider MUST NOT interact with the Principal and MUST NOT take
306 control of the user interface from the service provider. If "false," the identity provider MAY interact with
307 the user and MAY temporarily take control of the user interface for that purpose. If not specified, "true" is
308 presumed.

309 `ForceAuthn` [Optional]

310 Controls whether the identity provider authenticates the Principal regardless of whether the Principal is
311 already authenticated. This element is specified only when <IsPassive> is "false." If <ForceAuthn>
312 is "true," specifies that the identity provider MUST always authenticate the Principal, regardless of whether
313 the Principal is presently authenticated. If "false," specifies that the identity provider MUST re-authenticate
314 the user only if the Principal is not presently authenticated. If not specified, "false" is presumed.

-
- 315 ProtocolProfile [Optional]
316 The protocol profile that the requester wishes to use for the response. If the element is not specified, the
317 default protocol profile is *http://projectliberty.org/profiles/brws-art*, defined in [\[LibertyBindProf\]](#).
- 318 AssertionConsumerServiceID [Optional]
319 Used to direct the identity provider to use a specific assertion consumer service URL at the service provider.
320 It references an element in the provider's metadata with a matching id attribute.
- 321 RequestAuthnContext [Optional]
322 Information describing which authentication context the requester desires the identity provider to use in
323 authenticating the Principal.
- 324 RelayState [Optional]
325 This contains state information that will be relayed back in the response. This data SHOULD be integrity-
326 protected by the request author and MAY have other protections placed on it by the request author. An
327 example of such protection is confidentiality. Note that the actual requested resource URL at the service
328 provider SHOULD NOT be directly placed in this element unless the attendant privacy considerations do not
329 apply.
- 330 Scoping [Optional]
331 Specifies any preferences on the number and specific identifiers of additional identity providers through which
332 the authentication request may be proxied. See [Section 3.2.2.7](#) for rules regarding the proxying of identity
333 providers.
334 The requester may also choose not to include this element, in which case, the recipient of the message MAY
335 act as a proxy, according to the rules in [Section 3.2.2.7](#)
- 336 consent [Optional]
337 Indicates whether or not consent has been obtained from a user in sending this message.
- 338 The <RequestAuthnContext> element may contain the following elements, the first two of which are mutually
339 exclusive:
- 340 AuthnContextClassRef [Optional]
341 The ordered set of authentication context class references the requester desires the identity provider to use in
342 authenticating the Principal.
- 343 AuthnContextStatementRef [Optional]
344 The ordered set of exact authentication statements the requester desires the identity provider to use in
345 authenticating the Principal.
- 346 AuthnContextComparison [Optional]
347 If set to "exact", then the identity provider is asked to match at least one of the specified <AuthnContext>
348 elements exactly. This can also be set to "minimum", which asks that the identity provider use a context that
349 he feels is at least as good as any specified in the <AuthnContext> or "better", which means that the they
350 can use any context better than any that were supplied. If not specified, this is assumed to be "exact".
- 351 The <Scoping> element may contain the following elements:
- 352 ProxyCount [Optional]
353 The upper limit on the number of proxying steps the requester wishes to specify for this authentication
354 request.
- 355 IDPList [Optional]
356 An ordered list of identity providers which the requester prefers to use in authenticating the Principal. This
357 list is a suggestion only, and may be ignored or added to by the recipient of the message.

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

```

359 <xs:element name="AuthnRequest" type="AuthnRequestType" />
360 <xs:complexType name="AuthnRequestType">
361   <xs:complexContent>
362     <xs:extension base="samlp:RequestAbstractType">
363       <xs:sequence>
364         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
365         <xs:element ref="ProviderID" />
366         <xs:element ref="AffiliationID" minOccurs="0" />
367         <xs:element ref="NameIDPolicy" minOccurs="0" />
368         <xs:element name="ForceAuthn" type="xs:boolean" minOccurs="0" />
369         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0" />
370         <xs:element ref="ProtocolProfile" minOccurs="0" />
371         <xs:element name="AssertionConsumerServiceID" type="xs:string" minOccurs="0" />
372         <xs:element ref="RequestAuthnContext" minOccurs="0" />
373         <xs:element ref="RelayState" minOccurs="0" />
374         <xs:element ref="Scoping" minOccurs="0" />
375       </xs:sequence>
376       <xs:attribute ref="consent" use="optional" />
377     </xs:extension>
378   </xs:complexContent>
379 </xs:complexType>
380 <xs:simpleType name="NameIDPolicyType">
381   <xs:restriction base="xs:string">
382     <xs:enumeration value="none" />
383     <xs:enumeration value="onetime" />
384     <xs:enumeration value="federated" />
385     <xs:enumeration value="any" />
386   </xs:restriction>
387 </xs:simpleType>
388 <xs:element name="NameIDPolicy" type="NameIDPolicyType" />
389 <xs:simpleType name="AuthnContextComparisonType">
390   <xs:restriction base="xs:string">
391     <xs:enumeration value="exact" />
392     <xs:enumeration value="minimum" />
393     <xs:enumeration value="better" />
394   </xs:restriction>
395 </xs:simpleType>
396 <xs:complexType name="ScopingType">
397   <xs:sequence>
398     <xs:element name="ProxyCount" type="xs:nonNegativeInteger" minOccurs="0" />
399     <xs:element ref="IDPList" minOccurs="0" />
400   </xs:sequence>
401 </xs:complexType>
402 <xs:element name="Scoping" type="ScopingType" />
403 <xs:element name="RelayState" type="xs:string" />
404 <xs:element name="ProtocolProfile" type="xs:anyURI" />
405 <xs:element name="RequestAuthnContext">
406   <xs:complexType>
407     <xs:sequence>
408       <xs:choice>
409         <xs:element name="AuthnContextClassRef" type="xs:anyURI" maxOccurs="unbounded" />
410         <xs:element name="AuthnContextStatementRef" type="xs:anyURI" maxOccurs="unbounded" />
411       </xs:choice>
412     </xs:sequence>
413     <xs:element name="AuthnContextComparison" type="AuthnContextComparisonType" minOccurs="0" />
414   </xs:complexType>
415 </xs:element>
416 </xs:complexType>
417 </xs:element>
418
419
420

```

421 3.2.1.2. Example

```
422
423     <lib: AuthnRequest RequestID="RPCUk 211+GVz+t11LURp51oFvJXk"
424     MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
425     IssueInstant="2001-12-17T21:42:4Z" xmlns:lib="urn:liberty:iff:2003-08">
426     <ds:Signature> ... </ds:Signature>
427     <lib:ProviderID>http://ServiceProvider.com</lib:ProviderID>
428     <lib:NameIDPolicy>federate</lib:NameIDPolicy>
429     <lib:ForceAuthn>false</lib:ForceAuthn>
430     <lib:IsPassive>false</lib:IsPassive>
431     <lib:ProtocolProfile>http://projectliberty.org/profiles/brws-post</lib:ProtocolProfile
432 >
433     <lib:RequestAuthnContext>
434     <lib:AuthnContextClassRef>http://projectliberty.org/schemas/
435     authctx/classes/Password-ProtectedTransport</lib:AuthnContextClassRef>
436     <lib:AuthnContextComparison>exact</lib:AuthnContextComparison>
437 </lib:RequestAuthnContext>
438     <lib:RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</lib:RelayState>
439     <lib:Scoping>
440     <lib:ProxyCount>1</lib:ProxyCount>
441 </lib:Scoping>
442 </lib:AuthnRequest>
```

443 3.2.2. Response

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

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

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

451 If the <NameIDPolicy> element is omitted or "none", and if the service provider registered a name identifier
452 for the Principal, the <saml:NameIdentifier> element in the <saml:Subject> element **MUST** be the service
453 provider-provided name identifier for the Principal. Otherwise, <saml:NameIdentifier> **MUST** be the most
454 current name identifier supplied by the identity provider. The <IDPProvidedNameIdentifier> **MUST** contain
455 the most recent name identifier supplied by the identity provider. In either case, the Format attribute **MUST** be
456 *urn:liberty:iff:nameid:federated*.

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

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

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

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

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

468 Identity providers **MUST** include a <saml:AudienceRestrictionCondition> element that specifies the intended
469 consumers of the assertion. One <saml:Audience> element **MUST** be set to the intended recipient's ProviderID.

470 The recipient **MUST** validate that it is the intended viewer before using the assertion. The assertion **MAY** contain
471 additional `<saml:Audience>` elements that specify other intended relying parties.

472 Identity providers **MAY** include a `SessionIndex` attribute in resulting authentication statements, which is used to
473 aid the identity provider in managing multiple sessions with the Principal. If the identity provider includes this
474 `SessionIndex` attribute, subsequent messages from the service provider to the identity provider that are session-
475 dependent **MUST** include this `SessionIndex` attribute.

476 Identity providers **MAY** include other types of statements in the assertion(s) returned, depending on agreements
477 between providers and other specifications that provide additional functionality. Any such statements that include
478 a name identifier representing the Principal **MUST** be consistent with the identification semantics dictated by the
479 `<NameIDPolicy>` element. This is particularly relevant if the "onetime" policy is in effect; a temporary identifier or
480 an otherwise obfuscated and protected value **MUST** be used.

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

484 **3.2.2.1. Element `<AuthnResponse>`**

485 The type `AuthnResponseType` is extended from `samlp:ResponseType`.

486 The response contains the following elements:

487 `Extension` [Optional]

488 Optional container for protocol extensions established by agreement between providers.

489 `ProviderID` [Required]

490 The identity provider's unique identifier.

491 `RelayState` [Optional]

492 This contains state information being relayed.

493 `consent` [Optional]

494 Indicates whether or not consent has been obtained from a user in sending this message.

495 The schema fragment is as follows:

```
496 <xs:element name="AuthnResponse" type="AuthnResponseType"/>
497 <xs:complexType name="AuthnResponseType">
498   <xs:complexContent>
499     <xs:extension base="samlp:ResponseType">
500       <xs:sequence>
501         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
502         <xs:element ref="ProviderID"/>
503         <xs:element ref="RelayState" minOccurs="0"/>
504       </xs:sequence>
505       <xs:attribute ref="consent" use="optional"/>
506     </xs:extension>
507   </xs:complexContent>
508 </xs:complexType>
509
510
```

511 **3.2.2.2. Element `<AssertionType>`**

512 Authentication assertions provided in an `<AuthnResponse>` element **MUST** be of type **AssertionType**, which is
513 an extension of **saml:AssertionType**, so that the `RequestID` attribute from the original `<AuthnRequest>` **MAY**
514 be included in the `InResponseTo` attribute in the `<Assertion>` element. This is done because it is not required that

515 the <AuthnResponse> element itself be signed. Instead, the individual <Assertion> elements contained MUST
516 each be signed.

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

520 The schema fragment is as follows:

```
521 <xs:element name="Assertion" type="AssertionType" substitutionGroup="saml:Assertion" />
522 <xs:complexType name="AssertionType">
523   <xs:complexContent>
524     <xs:extension base="saml:AssertionType">
525       <xs:attribute name="InResponseTo" type="xs:NCName" use="optional"/>
526     </xs:extension>
527   </xs:complexContent>
528 </xs:complexType>
529
530
```


531 3.2.2.3. SubjectType and Related Types

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

534 Liberty name identifier elements are directly based on **saml:NameIdentifierType**. Liberty ID-FF use of SAML-
535 defined attributes is as follows - it should be noted that NameQualifier and Format attributes are required in Liberty
536 messages, but are optional in SAML (see [SAMLCore1]):

537 NameQualifier [Required]

538 Generally used to indicate the unique identifier of the service provider or affiliation group for or by whom the
539 name identifier was created. Unless otherwise specified, the service provider's or affiliation group's unique
540 identifier **MUST** be placed in this attribute to disambiguate the identifier from any other identity federations
541 the principal may have with the identity provider.

542 Format [Required]

543 Indicates the format, semantics, and processing rules associated with the identifier. One of these four values
544 **MUST** be present:

545 *urn:liberty:iff:nameid:federated*

546 Used for identifiers communicated on behalf of Principals that have federated their identity.

547 *urn:liberty:iff:nameid:one-time*

548 Used for identifiers with anonymous, single-use semantics communicated on behalf of
549 Principals that have not federated or wish to act anonymously.

550 *urn:liberty:iff:nameid:encrypted*

551 Used for identifiers that have been encrypted for use only by a specific provider. Such an
552 identifier **MUST** be a base-64 encoded <EncryptedNameIdentifier> element, defined
553 below. See the Name Identifier Encryption Profile in [LibertyBindProf] for more
554 information.

555 *urn:liberty:iff:nameid:entityID*

556 Used for identifiers that identify a Liberty provider or affiliation group. This may be used
557 to indicate that the subject indicated in an authentication statement is a Liberty provider or
558 affiliation group.

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

561 **xenc:EncryptedData** [Required]

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

567 **xenc:EncryptedKey** [Optional]

568 A wrapped symmetric encryption key used to encrypt the <EncryptableNameIdentifier> element, per
569 [xmlenc-core].

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

573 Format [Required]
574 MUST contain a value of *urn:liberty:iff:nameid:federated* (of the formats defined in this specification, only
575 *federated* name identifiers sometimes require encryption).

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

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

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

589 The schema fragment is as follows:

```
590 <xs:complexType name="SubjectType">
591   <xs:complexContent>
592     <xs:extension base="saml:SubjectType">
593       <xs:sequence>
594         <xs:element ref="IDPProvidedNameIdentifier"/>
595       </xs:sequence>
596     </xs:extension>
597   </xs:complexContent>
598 </xs:complexType>
599 <xs:element name="Subject" type="SubjectType" substitutionGroup="saml:Subject"/>
600 <xs:element name="EncryptableNameIdentifier" type="EncryptableNameIdentifierType"
601   substitutionGroup="saml:NameIdentifier"/>
602 <xs:complexType name="EncryptableNameIdentifierType">
603   <xs:simpleContent>
604     <xs:extension base="saml:NameIdentifierType">
605       <xs:attribute name="IssueInstant" type="xs:dateTime"/>
606       <xs:attribute name="Nonce" type="xs:string"/>
607     </xs:extension>
608   </xs:simpleContent>
609 </xs:complexType>
610 <xs:element name="EncryptedNameIdentifier" type="EncryptedNameIdentifierType"/>
611 <xs:complexType name="EncryptedNameIdentifierType">
612   <xs:sequence>
613     <xs:element ref="xenc:EncryptedData"/>
614     <xs:element ref="xenc:EncryptedKey" minOccurs="0"/>
615   </xs:sequence>
616 </xs:complexType>
617
618
619
```

620 3.2.2.4. Type AuthenticationStatementType

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

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

627 ReauthenticateOnOrAfter [Optional]
 628 The time at, or after which the service provider reauthenticates the Principal with the identity provider (as
 629 required in the [Section 3.2.2.6](#) [22]processing rules for this protocol).

630 SessionIndex [Optional]
 631 Indexes the particular session between the Principal and the identity provider under which this authentication
 632 statement is being issued. This value SHOULD be a small, positive integer but may be any string of text.
 633 However, this value MUST NOT be a globally unique value for the Principal's session at the identity provider.
 634 When an <AuthnContext> element is specified, the saml:AuthenticationMethod attribute on the
 635 <saml:AuthenticationStatement> MUST be <http://projectliberty.org/schemas/authctx/2002/05>.
 636 When the service provider is processing a <saml:AuthenticationStatement> of type
 637 **lib:AuthenticationStatementType** and the saml:AuthenticationMethod attribute is
 638 <http://projectliberty.org/schemas/authctx/2002/05>, the service provider MUST refer to the <AuthnContext>
 639 element and ignore the saml:AuthenticationMethod attribute.

640 The schema fragment is as follows:

```

641 <xs:element name="AuthenticationStatement" type="AuthenticationStatementType" substit
642 tionGroup="saml:Statement"/>
643 <xs:complexType name="AuthenticationStatementType">
644   <xs:complexContent>
645     <xs:extension base="saml:AuthenticationStatementType">
646       <xs:sequence>
647         <xs:element ref="AuthnContext" minOccurs="0"/>
648       </xs:sequence>
649       <xs:attribute name="ReauthenticateOnOrAfter" type="xs:dateTime" use="optional"/>
650       <xs:attribute name="SessionIndex" type="xs:string" use="optional"/>
651     </xs:extension>
652   </xs:complexContent>
653 </xs:complexType>
654 <xs:element name="AuthnContext">
655   <xs:complexType>
656     <xs:sequence>
657       <xs:element name="AuthnContextClassRef" type="xs:anyURI" minOccurs="0"/>
658       <xs:choice>
659         <xs:element ref="ac:AuthenticationContextStatement"/>
660         <xs:element name="AuthnContextStatementRef" type="xs:anyURI"/>
661       </xs:choice>
662     </xs:sequence>
663   </xs:complexType>
664 </xs:element>
665
666
```

667 3.2.2.5. Example

```

668 <lib:AuthnResponse ResponseID="hhuujalbc744hGJn5Q9A5yvEIgS"
669   InResponseTo="Zon3WjJ2KL7j+bJu7MuIr 4Pt2go5" MajorVersion="1" MinorVersion="2"
670   consent="urn:liberty:consent:obtained" IssueInstant="2002-10-31T21:55:41Z">
671   <samlp:Status>
672     <samlp:StatusCode Value="samlp:Success"/>
673   </samlp:Status>
674   <lib:Assertion MajorVersion="1" MinorVersion="2"
675     AssertionID="e06e5a28-bc80-4ba6-9ecb-712949db686e"
676     Issuer="http://IdentityProvider.com" IssueInstant="2001-12-17T09:30:47Z"
677     InResponseTo="4e7c3772-4fa4-4a0f-99e8-7d719ff6067c">
678     <saml:Conditions NotBefore="2001-12-17T09:30:47Z" NotOnOrAfter="2001-1
679
```

```

680 2-17T09:35:47Z">
681         <saml:AudienceRestrictionCondition>
682             <saml:Audience>http://ServiceProvider.com</saml:Audience>
683         </saml:AudienceRestrictionCondition>
684     </saml:Conditions>
685     <lib:AuthenticationStatement AuthenticationInstant="2001-12-17T09:30:47Z"
686         SessionIndex="3" ReauthenticateOnOrAfter="2001-12-17T11:30:47Z"
687         AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
688         <lib:Subject>
689             <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
690                 Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-
691 4c68-be35-cc9e7db39e2b</saml:NameIdentifier>
692             <saml:SubjectConfirmation>
693                 <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer</saml:Conf
694 irmationMethod>
695             </saml:SubjectConfirmation>
696             <IDPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
697                 Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-
698 4c68-be35-cc9e7db39e2b</IDPProvidedNameIdentifier>
699         </lib:Subject>
700     </lib:AuthenticationStatement>
701     <ds:Signature>...</ds:Signature>
702 </lib:Assertion>
703 <lib:ProviderID>http://IdentityProvider.com</lib:ProviderID>
704 <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
705 </lib:AuthnResponse>

```

707 3.2.2.6. Processing Rules

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

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

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

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

717 A `<NameIDPolicy>` of *any*.

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

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

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

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

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

- 730 If the <AssertionConsumerServiceID> element is not provided, then the identity provider MUST use the default
731 URL (the <AssertionConsumerServiceURL> with an `isDefault` attribute of "true").
- 732 <ds:Signature>, if present, MUST be the signature of the service provider as specified by the <ProviderID>.
- 733 If the requesting provider's <AuthnRequestsSigned> metadata element is "true", then any request messages it
734 generates MUST be signed. If an unsigned request is received, then the provider MUST return an error with a second-
735 level <samlp:StatusCode> of *lib:UnsignedAuthnRequest*.
- 736 If <IsPassive> is "true," the identity provider MUST NOT interact with the Principal and MUST NOT take control
737 of the user interface (if applicable).
- 738 The identity provider MUST attempt to authenticate the Principal if <ForceAuthn> is "true," regardless of whether
739 the Principal is presently authenticated, unless <IsPassive> is "true."
- 740 Success in authenticating the Principal is indicated by a status code of *samlp:Success* and a signed assertion containing
741 at least one statement of type **lib:AuthenticationStatementType** representing the Principal's authentication
742 information. Other types of statements may also be included, as defined by providers and other specifications.
- 743 Failure to authenticate the Principal is indicated by a status code other than *samlp:Success*. For failures, assertions
744 MUST NOT be included in the <AuthnResponse>.
- 745 <AffiliationID>, if present, MUST be the unique identifier of a known affiliation group with which the identity
746 provider has an established relationship, and of which the requesting provider is a member. If present, identity
747 providers MUST establish and resolve federations based on the specified affiliation, not the requesting provider. In
748 addition, identity providers MAY retrieve information regarding the other members of the affiliation group by querying
749 metadata (see [\[LibertyMetadata\]](#)) and present a list of members of the affiliation group to the Principal.
- 750 The following rules apply to the selection of name identifiers and the federation process:
- 751 If the <NameIDPolicy> element is omitted or "none", then the identity provider MUST return the name identifier(s)
752 corresponding to the federation that exists between the identity provider and the requesting provider or affiliation
753 group for the Principal. If no such federation exists, then an error with a second-level <samlp:StatusCode> of
754 *lib:FederationDoesNotExist* MUST be returned to the provider.
- 755 If the <NameIDPolicy> element is "onetime", then the <saml:NameIdentifier> element in the <saml:Subject>
756 element MUST be a temporary, one-time-use identifier for the Principal, with a `Format` attribute of
757 *urn:liberty:iff:nameid:one-time*.
- 758 If <NameIDPolicy> is *federated*, and if the Principal consents, then the identity provider MAY federate the Principal's
759 identity with the requesting provider (or the affiliation group if <AffiliationID> is present). If the identity provider
760 already has a previous federation on record for the Principal's identity at the requesting provider or affiliation group
761 (such as when a provider previously issued a <FederationTerminationNotification> which was not received
762 by the identity provider), then the identity provider SHOULD treat the request as if <NameIDPolicy> were *none*.
- 763 If <NameIDPolicy> is *any*, then the rules above for the values of *federated* and *onetime* MUST be followed, in that
764 order. Thus, a new federation may be created, an existing federation used, or a temporary identifier generated.
- 765 When including a Principal's federated identity in the response, the <Subject> element MUST include a
766 <saml:NameIdentifier>, containing the most recent identifier set by the service provider or affiliation group for
767 that Principal at the identity provider. If no such value has been provided, then the identifier set by the identity
768 provider MUST be used. In either case, the <Subject> MUST include an <IDPProvidedNameIdentifier>
769 containing the identifier set by the identity provider.
- 770 The <Subject> MUST contain a <SubjectConfirmation> element in accordance with the SSO profile used to
771 return the response (see [\[SAMLBind11\]](#) and [\[LibertyBindProf\]](#)).

- 772 When federating or in the case of a temporary value, the identity provider MUST adhere to the following rules in
773 generating the name identifier:
- 774 The name identifier MUST be unique across all Principals in the scope of that requester-identity provider relationship.
- 775 The name identifier for the specific Principal MUST be unique across all providers with which an identity federation
776 exists with the identity provider.
- 777 The identity provider MUST respond using the specified `<ProtocolProfile>`.
- 778 If `<RelayState>` contains a value, the identity provider MUST include this value in unmodified form in the
779 `<RelayState>` element of the returned authentication assertion.
- 780 The *InResponseTo* attribute in all generated `<Assertion>` elements in the `<AuthnResponse>` element MUST be set
781 to the value of the `RequestID` attribute in the corresponding `<AuthnRequest>` element. If there is no such request
782 because the identity provider is initiating the response on its own, then the attribute MUST NOT be included.
- 783 Additionally, if the `<RequestAuthnContext>` element is specified, the identity provider MUST authenticate the
784 Principal according to the following rules:
- 785 If one or more `<AuthnContextClassRef>` or `<AuthnContextStatementRef>` elements are included, then the
786 resulting authentication statement in the assertion (if any) MUST contain an authentication statement that conforms
787 to the class or statement specified. Additionally, the set of supplied elements MUST be evaluated as an ordered set,
788 where the first element is the most preferred authentication context class or statement. If none of the specified classes
789 or statements can be satisfied, the identity provider MUST not include an authentication statement in the resulting
790 assertion.
- 791 Additionally, if an `<AuthnContextComparison>` element is supplied, and one or more `<AuthnContextStatementRef>`
792 or `<AuthnContextClassRef>` elements are included, then the resulting authentication statement in the assertion (if
793 any) MUST follow the rule specified in the `<AuthnContextComparison>` element. If this requirement cannot be
794 satisfied, the identity provider MUST NOT include an authentication statement in the resulting assertion.
- 795 If `<AuthnContextComparison>` is specified and set to *exact*, then the resulting authentication statement in the
796 assertion (if any) MUST be the exact match of at least one of the authentication contexts specified.
- 797 If `<AuthnContextComparison>` is specified and set to *minimum*, then the resulting authentication statement in the
798 assertion (if any) MUST be at least as strong (as deemed by the identity provider) as one of the authentication contexts
799 specified.
- 800 If `<AuthnContextComparison>` is specified and set to *better*, then the resulting authentication statement in the
801 assertion (if any) MUST be stronger (as deemed by the identity provider) than any specified in the supplied
802 authentication contexts.
- 803 If the identity provider wishes to rely on a second identity provider as the source of the Principal's authentication, then
804 the provider MUST follow the rules specified in [Section 3.2.2.7](#).
- 805 If the requesting provider attempts to federate a Principal's identity with an identity provider, but another Principal's
806 identity at the same requesting provider is already federated with the same identity provider, it will receive the other
807 Principal's established name identifier in the `<AuthnResponse>`, rather than a new random one. The requesting
808 provider MUST detect this error and handle it appropriately without leaving either Principal's identity at the provider
809 in an unusable state.
- 810 The resulting authentication statement in the assertion by the identity provider MAY contain a
811 `ReauthenticateOnOrAfter` attribute. If this attribute is included, the service provider MUST send a new
812 `<AuthnRequest>` for the Principal to the identity provider at the next point of interaction with the Principal on
813 or after the time specified by the `ReauthenticateOnOrAfter` attribute. It is then up to the identity provider to
814 authenticate the user.

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

817 **3.2.2.7. Dynamic Proxying of Identity Providers**

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

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

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

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

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

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

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

833 When creating the new authentication request:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

871 **3.2.2.8. Active Intermediaries**

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

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

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

884 Specify whether the `<AuthnRequest>` element sent from the service provider to the identity provider via the
885 intermediary is wrapped in an `<AuthnRequestEnvelope>`. See section 3.2.4.

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

888 **3.2.2.8.1. Processing Rules for Active Intermediaries**

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

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

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

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

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

904 • The intermediary MUST remove the enveloping <AuthnResponseEnvelope> before forwarding the
905 <AuthnResponse> element to the service provider.

906 • The intermediary MUST send <AuthnResponse> messages received from the identity provider to the service
907 provider using the <AssertionConsumerServiceURL> contained within the <AuthnResponseEnvelope> sent
908 by the identity provider.

909 3.2.2.9. Status Code Values for Error Conditions

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

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

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

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

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

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

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

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

927 3.2.3. Request Envelope

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

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

934 3.2.3.1. Element <AuthnRequestEnvelope>

935 The authentication request envelope contains the following elements:

936 `Extension` [Optional]

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

940 `AuthnRequest` [Required]

941 The authentication request contained within the envelope.

942 `ProviderID` [Required]

943 The requester's `ProviderID`.

944 `ProviderName` [Optional]

945 The human-readable name of the requester.

946 `AssertionConsumerServiceURL` [Required]

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

949 `IDPList` [Optional]

950 A list of identity providers, from which, one may be chosen to service the authentication request.

951 `IsPassive` [Optional]

952 If "true," specifies that any intermediary between the service provider and identity provider MUST NOT
953 interact with the Principal. If not specified, "true" is presumed.

954 The schema fragment is as follows:

```
955 <xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType" />
956 <xs:complexType name="AuthnRequestEnvelopeType">
957   <xs:complexContent>
958     <xs:extension base="RequestEnvelopeType">
959       <xs:sequence>
960         <xs:element ref="AuthnRequest" />
961         <xs:element ref="ProviderID" />
962         <xs:element name="ProviderName" type="xs:string" minOccurs="0" />
963         <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI" />
964         <xs:element ref="IDPList" minOccurs="0" />
965         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0" />
966       </xs:sequence>
967     </xs:extension>
968   </xs:complexContent>
969 </xs:complexType>
970 <xs:complexType name="RequestEnvelopeType">
971   <xs:sequence>
972     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
973   </xs:sequence>
974 </xs:complexType>
975
976
```

977 3.2.3.2. Element `<IDPList>`

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

- 981 IDPList [Optional]
982 The container element for an IDP List.
- 983 IDPEntries [Required]
984 Contains a list of identity provider entries.
- 985 IDPEntry [Required]
986 Describes an identity provider that the service provider supports.
- 987 ProviderID [Required]
988 The identity provider's unique identifier.
- 989 ProviderName [Optional]
990 The identity provider's human-readable name.
- 991 Loc [Optional]
992 The identity provider's URI, to which authentication requests may be sent. If present, this MUST be set
993 to the value of the identity provider's <SingleSignOnService> element, obtained from their metadata
994 ([\[LibertyMetadata\]](#)).
- 995 GetComplete [Optional]
996 If the identity provider list is not complete, this element may be included with a URI that points to where the
997 complete list can be retrieved.

998 The schema fragment is as follows:

```
999     <xs:element name="IDPList" type="IDPListType"/>
1000     <xs:complexType name="IDPListType">
1001       <xs:sequence>
1002         <xs:element ref="IDPEntries"/>
1003         <xs:element ref="GetComplete" minOccurs="0"/>
1004       </xs:sequence>
1005     </xs:complexType>
1006     <xs:element name="IDPEntry">
1007       <xs:complexType>
1008         <xs:sequence>
1009           <xs:element ref="ProviderID"/>
1010           <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
1011           <xs:element name="Loc" type="xs:anyURI"/>
1012         </xs:sequence>
1013       </xs:complexType>
1014     </xs:element>
1015     <xs:element name="IDPEntries">
1016       <xs:complexType>
1017         <xs:sequence>
1018           <xs:element ref="IDPEntry" maxOccurs="unbounded"/>
1019         </xs:sequence>
1020       </xs:complexType>
1021     </xs:element>
1022     <xs:element name="GetComplete" type="xs:anyURI"/>
1023
1024
```

1025 3.2.3.3. Example

```
1026     <AuthnRequestEnvelope>
1027       <AuthnRequest> ... </AuthnRequest>
1028       <ProviderID>http://ServiceProvider.com</ProviderID>
1029       <ProviderName>Service Provider X</ProviderName>
1030       <AssertionConsumerServiceURL>http://ServiceProvider.com/lecp_assertion_consume
1031       r</AssertionConsumerServiceURL>
1032       <IDPList>
```

```

1034         <IDPEntries>
1035         <IDPEntry>
1036         <ProviderID>http://IdentityProvider.com</ProviderID>
1037         <ProviderName>Identity Provider X</ProviderName>
1038         <Loc>http://www.IdentityProvider.com/liberty/so</Loc>
1039     </IDPEntry>
1040 </IDPEntries>
1041     <GetComplete>https://ServiceProvider.com/idplist?id=604bel36-fe91-441e-afb8-f887
1042 48ae3b8b </GetComplete>
1043 </IDPList>
1044     <IsPassive>0</IsPassive>
1045 </AuthnRequestEnvelope>

```

1046 3.2.4. Response Envelope

1047 As with the <AuthnRequest> element, some profiles MAY wrap the <AuthnResponse> element in an envelope.
1048 This envelope allows for extra processing by an intermediary (such as a user agent or proxy) between the identity
1049 provider and the service provider. Applicable processing rules are given in section 3.2.3.3.1. Note that the envelope
1050 is for consumption by the intermediary and is removed prior to the forwarding of the enveloped <AuthnResponse>
1051 element to the service provider.

1052 3.2.4.1. Element <AuthnResponseEnvelope>

1053 The authentication response envelope contains the following elements:

1054 Extension [Optional]

1055 Optional container for protocol extensions established by agreement between service providers and interme-
1056 diaries. Implementers should note that this element may not contain content from the core Liberty namespace
1057 (which is prevented at the schema level by requiring namespace="##other").

1058 AuthnResponse [Required]

1059 The enveloped authentication response.

1060 AssertionConsumerServiceURL [Required]

1061 The service provider's URL where the authentication response should be sent. This element's value SHOULD
1062 be obtained from the element of the same name in the service provider's metadata (see [\[LibertyMetadata\]](#)).

1063 The schema fragment is as follows:

```

1064 <xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType"/>
1065 <xs:complexType name="AuthnResponseEnvelopeType">
1066     <xs:complexContent>
1067         <xs:extension base="ResponseEnvelopeType">
1068             <xs:sequence>
1069                 <xs:element ref="AuthnResponse"/>
1070                 <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
1071             </xs:sequence>
1072         </xs:extension>
1073     </xs:complexContent>
1074 </xs:complexType>
1075 <xs:complexType name="ResponseEnvelopeType">
1076     <xs:sequence>
1077         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1078     </xs:sequence>
1079 </xs:complexType>
1080
1081

```

1082 3.2.4.2. Example

```
1083
1084         <AuthnResponseEnvelope>
1085             <AuthnResponse> . . . </AuthnResponse>
1086             <AssertionConsumerServiceURL>
1087                 http://ServiceProvider.com/lecp_assertion_consumer
1088             </AssertionConsumerServiceURL>
1089         </AuthnResponseEnvelope>
```

1090 **3.3. Name Registration Protocol**

1091 During federation, the identity provider generates an opaque handle that serves as the initial name identifier that both
1092 the service provider and the identity provider use in referring to the Principal when communicating with each other.
1093 This name identifier is termed the `<IDPProvidedNameIdentifier>`.

1094 Subsequent to federation, the service provider *MAY* register a different opaque handle with the identity provider. This
1095 opaque handle is termed the `<SPProvidedNameIdentifier>`. Until the service provider registers a different name,
1096 the identity provider will use `<IDPProvidedNameIdentifier>` to refer to the Principal when communicating with
1097 the service provider.

1098 After a service provider's name registration, the identity provider *MUST* use the `<SPProvidedNameIdentifier>`
1099 for `<saml:NameIdentifier>` elements when communicating to the service provider about the Principal.
1100 The service provider *MUST* use the current (most recently supplied) `<IDPProvidedNameIdentifier>` for
1101 `<saml:NameIdentifier>` elements when communicating to the identity provider about the Principal.

1102 Either the service provider or the identity provider *MAY* register a new name identifier for a Principal with each
1103 other at any time following federation. The name identifiers specified by providers *SHOULD* adhere to the following
1104 guidelines:

1105 • The name identifier *SHOULD* be unique across the identity providers with which the Principal's identity is
1106 federated.

1107 • The name identifier *SHOULD* be unique within the group of name identifiers that have been registered with the
1108 identity provider by this service provider.

1109 **3.3.1. Request**

1110 To register a `<SPProvidedNameIdentifier>` with an identity provider, the service provider sends a
1111 `<RegisterNameIdentifierRequest>` message.

1112 The same `<RegisterNameIdentifierRequest>` message may be sent by an identity provider, seeking to change
1113 the `<IDPProvidedNameIdentifier>` stored by the service provider.

1114 The `<RegisterNameIdentifierRequest>` message *SHOULD* be signed.

1115 **3.3.1.1. Element `<RegisterNameIdentifierRequest>`**

1116 The elements of the message are as follows:

1117 `Extension` [Optional]

1118 Optional container for protocol extensions established by agreement between providers.

1119 `ProviderID` [Required]

1120 The provider's identifier.

1121 `IDPProvidedNameIdentifier` [Required]

1122 The name identifier the service provider should use when communicating with the identity provider.

1123 `SPProvidedNameIdentifier` [Required]

1124 The name identifier the identity provider should use when communicating to the service provider.

1125 OldProvidedNameIdentifier [Required]
 1126 In the case of either provider choosing to request a change of provided name identifiers, this element
 1127 holds the previous version. For a service provider making their first name change following federation,
 1128 the <OldProvidedNameIdentifier> will contain the current <IDPProvidedNameIdentifier>. The
 1129 <SPPProvidedNameIdentifier> will contain the new name that the service provider wishes the identity
 1130 provider to use.

1131 RelayState [Optional]
 1132 This contains state information that will be relayed back in the response. This data SHOULD be integrity-
 1133 protected by the request author and MAY have other protections placed on it by the request author. An
 1134 example of such protection is confidentiality.

1135 The schema fragment is as follows:

```

1136 <xs:element name="RegisterNameIdentifierRequest" type="RegisterNameIdentifierRequestType"/>
1137 <xs:complexType name="RegisterNameIdentifierRequestType">
1138   <xs:complexContent>
1139     <xs:extension base="samlp:RequestAbstractType">
1140       <xs:sequence>
1141         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1142         <xs:element ref="ProviderID"/>
1143         <xs:element ref="IDPProvidedNameIdentifier"/>
1144         <xs:element ref="SPPProvidedNameIdentifier"/>
1145         <xs:element ref="OldProvidedNameIdentifier"/>
1146         <xs:element ref="RelayState" minOccurs="0"/>
1147       </xs:sequence>
1148     </xs:extension>
1149   </xs:complexContent>
1150 </xs:complexType>
1151 <xs:element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1152 <xs:element name="SPPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1153 <xs:element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1154
1155
    
```

1156 3.3.1.2. Example

```

1157
1158 <RegisterNameIdentifierRequest RequestID="eb20e77f-d982-44f9-936e-dd135bf437d4"
1159   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">
1160   <ds:Signature>...</ds:Signature>
1161   <ProviderID>http://ServiceProvider.com</ProviderID>
1162   <IDPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
1163     Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be35-cc9e7db39e2b</IDPPr
1164   ovidedNameIdentifier>
1165   <SPPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
1166     Format="urn:liberty:iff:nameid:federated">e958019a</SPPProvidedNameIdentifier>
1167   <OldProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
1168     Format="urn:liberty:iff:nameid:federated">e895014a</OldProvidedNameIdentifier
1169   >
1170   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1171 </RegisterNameIdentifierRequest>
1172
    
```

1173 3.3.2. Response

1174 The recipient MUST respond with a <RegisterNameIdentifierResponse> message, which is of
 1175 type **StatusResponseType**. **StatusResponseType** is an extension of **samlp:ResponseType** and a
 1176 <samlp:Status> element and a <RelayState> may exist in the body.

1177 This message SHOULD be signed.

1178 3.3.2.1. Element <RegisterNameIdentifierResponse>

1179 The elements of the message are as follows:

1180 Extension [Optional]

1181 Optional container for protocol extensions established by agreement between providers.

1182 ProviderID [Required]

1183 The provider's unique identifier.

1184 Status [Required]

1185 The status of the request processing.

1186 RelayState [Optional]

1187 This element contains state information that will be relayed back in the response, if it has been supplied in
1188 the request.

1189 The schema fragment is as follows:

```
1190 <xs:element name="RegisterNameIdentifierResponse" type="StatusResponseType"/>
1191 <xs:complexType name="StatusResponseType">
1192   <xs:complexContent>
1193     <xs:extension base="samlp:ResponseAbstractType">
1194       <xs:sequence>
1195         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1196         <xs:element ref="ProviderID"/>
1197         <xs:element ref="samlp:Status"/>
1198         <xs:element ref="RelayState" minOccurs="0"/>
1199       </xs:sequence>
1200     </xs:extension>
1201   </xs:complexContent>
1202 </xs:complexType>
1203
1204
```

1205 3.3.2.2. Example

```
1206
1207 <RegisterNameIdentifierResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
1208
1209   InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
1210   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
1211   Recipient="http://ServiceProvider.com">
1212   <ds:Signature>...</ds:Signature>
1213   <ProviderID>http://ServiceProvider.com</ProviderID>
1214   <samlp:Status>
1215     <samlp:StatusCode Value="samlp:Success"/>
1216   </samlp:Status>
1217   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1218 </RegisterNameIdentifierResponse>
```

1219 3.3.3. Processing Rules

1220 The recipient MUST validate any signature present on the message. To be considered valid, the signature provided
1221 MUST be the signature of the <ProviderID> contained in the message.

1222 If the request includes an <IDPProvidedNameIdentifier> for which no federation exists between the service
1223 provider and the identity provider, the provider MUST respond with a <samlp:Status> element containing a
1224 second-level <samlp:StatusCode> of lib:FederationDoesNotExist. Otherwise, the identity provider MUST use
1225 <SPPProvidedNameIdentifier> when subsequently communicating to the service provider regarding this Principal.

1226 Either provider MAY choose to change their provided name identifier. In this case, the `<OldProvidedNameIdentifier>`
1227 should contain the previous version of their name identifier. When a service provider chooses to
1228 change their provided name identifier, the `<OldProvidedNameIdentifier>` should contain the current
1229 `<SPProvidedNameIdentifier>`. Note that when they first change their name, this will be equal to the
1230 `<IDPProvidedNameIdentifier>`. Similarly, when an identity provider wishes to change their provided name
1231 identifier, they will move the previous version to the `<OldProvidedNameIdentifier>` when sending this message.

1232 In all of the name identifier elements in the request and response messages of this protocol, if the Principal's identity
1233 federation is between the identity provider and an affiliation group in which the service provider is a member, then the
1234 `NameQualifier` attribute MUST contain the unique identifier of the affiliation group. Otherwise, it MUST contain
1235 the unique identifier of the service provider. This attribute MUST be used by the providers to identify the specific
1236 identity federation to be modified.

1237 Changes to these identifiers may take a potentially significant amount of time to propagate through the systems at both
1238 the sender and the receiver. Implementations MAY wish to allow each party to accept either identifier for some period
1239 of time following the successful completion of a name identifier change. Not doing so could result in the inability of
1240 the Principal to access resources.

1241 If `<RelayState>` contains a value, the recipient MUST include this value in unmodified form in the `<RelayState>`
1242 element of the response.

1243 **3.4. Federation Termination Notification Protocol**

1244 When the Principal terminates an identity federation between a service provider and an identity provider from the
1245 service provider, the service provider MUST send a `<FederationTerminationNotification>` message to the
1246 identity provider. The service provider is stating that it will no longer accept authentication assertions from the identity
1247 provider for the specified Principal.

1248 Likewise, when the Principal terminates an identity federation from the identity provider, the identity provider MUST
1249 send a `<FederationTerminationNotification>` message to the service provider. In this case, the identity
1250 provider is stating that it will no longer provide authentication assertions to the service provider for the specified
1251 Principal.

1252 This notification message is a one-way asynchronous message. Reasonable, best-effort delivery MUST be employed
1253 by all providers sending this message.

1254 **3.4.1. Message**

1255 The provider sends a `<FederationTerminationNotification>` to the provider with which it is terminating a
1256 federation.

1257 The `<FederationTerminationNotification>` message SHOULD be signed.

1258 **3.4.1.1. Element `<FederationTerminationNotification>`**

1259 The elements are as follows:

1260 `Extension` [Optional]

1261 Optional container for protocol extensions established by agreement between providers.

1262 `ProviderID` [Required]

1263 The identifier of the provider that is sending this message.

1264 **NameIdentifier [Required]**
1265 The name identifier of the Principal terminating federation. This name identifier **MUST** be equal to
1266 the <saml:NameIdentifier> element (and its included attributes) agreed upon earlier between the two
1267 communicating providers.

1268 **consent [Optional]**
1269 Indicates whether or not consent has been obtained from a user in sending this message.

1270 The schema fragment is as follows:

```
1271 <xs:element name="FederationTerminationNotification" type="FederationTerminationNotif
1272 icationType"/>
1273 <xs:complexType name="FederationTerminationNotificationType">
1274 <xs:complexContent>
1275 <xs:extension base="samlp:RequestAbstractType">
1276 <xs:sequence>
1277 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1278 <xs:element ref="ProviderID"/>
1279 <xs:element ref="saml:NameIdentifier"/>
1280 </xs:sequence>
1281 <xs:attribute ref="consent" use="optional"/>
1282 </xs:extension>
1283 </xs:complexContent>
1284 </xs:complexType>
1285
1286
```

1287 3.4.1.2. Example

```
1288
1289 <FederationTerminationNotification RequestID="e9c2-eb65-4bce-ab8f-4becdf229815"
1290 MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
1291 IssueInstant="2001-12-17T09:30:47Z">
1292 <ds:Signature>...</ds:Signature>
1293 <ProviderID>http://IdentityProvider.com</ProviderID>
1294 <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
1295 Format="urn:liberty:iff:nameid:federated">e958019a</saml:NameIdentifier>
1296 <RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1297 </FederationTerminationNotification>
```

1298 3.4.2. Processing Rules

1299 The receiving provider **MUST** validate any signature present on the message. The signature on the message **MUST** be
1300 the signature of the <ProviderID> contained in the message. If the signature is not valid, the provider **MUST** ignore
1301 the message.

1302 If a provider receives a federation termination notification message that refers to a federation that does not exist from
1303 the perspective of the provider, the provider **MUST** ignore the message. Otherwise, the provider **MAY** perform any
1304 maintenance with the knowledge that the federation has been terminated.

1305 A provider **MAY** choose to invalidate the session of a user for whom federation has been terminated.

1306 If the Principal's identity federation was between the identity provider and an affiliation group in which the service
1307 provider is a member, then the `NameQualifier` attribute **MUST** contain the unique identifier of the affiliation group.
1308 Otherwise, it **MUST** contain the unique identifier of the service provider. This attribute **MUST** be used by the providers
1309 to identify the specific identity federation being terminated.

1310 3.5. Single Logout Protocol

1311 The Single Logout Protocol provides a message exchange protocol by which all sessions authenticated by a particular
1312 identity provider are near-simultaneously terminated. The Single Logout Protocol is used either when a Principal logs
1313 out at a service provider or when the Principal logs out at an identity provider.

1314 When the Principal invokes the single logout process at a service provider, the service provider **MUST** send a
1315 <LogoutRequest> message to the identity provider that provided the authentication service for the session.

1316 When either the Principal invokes a logout at the identity provider or a service provider sends a logout request to
1317 the identity provider specifying that Principal, the identity provider **MUST** send a <LogoutRequest> message to
1318 each service provider to which it provided authentication assertions in the current session with the Principal, with the
1319 exception of the service provider that sent the <LogoutRequest> message to the identity provider.

1320 If the identity provider is proxying authentication from a second identity provider, then it **MUST** send
1321 a <LogoutRequest> to the proxied identity provider, unless the proxying provider has already received a
1322 <LogoutRequest> from the proxied provider.

1323 If the identity provider has provided authentication assertions on behalf of a Principal to a proxying identity provider,
1324 then it **MUST** send a <LogoutRequest> to that provider, unless the proxying provider has already received a
1325 <LogoutRequest> from the proxied provider.

1326 **3.5.1. Request**

1327 The <LogoutRequest> message indicates to the message receiver that a Principal's session was terminated. The
1328 message includes an optional <SessionIndex> element that **MUST** be specified if and only if the authentication
1329 statement in the assertion that the service provider used in establishing the session with the Principal contained a
1330 SessionIndex attribute. This message **SHOULD** be signed.

1331 **3.5.1.1. Element <LogoutRequest>**

1332 **Extension** [Optional]

1333 Optional container for protocol extensions established by agreement between providers. Implementers should
1334 note that this element may not contain content from the core Liberty namespace (which is prevented at the
1335 schema level by requiring namespace="##other").

1336 **NameIdentifier** [Required]

1337 The name identifier of the Principal that logged out. This name identifier **MUST** be equal to the
1338 <saml:NameIdentifier> element (including the equality of contained attributes) agreed upon between
1339 the two communicating providers.

1340 **ProviderID** [Required]

1341 The identifier of the provider that is making the request.

1342 **SessionIndex** [Optional]

1343 The session index specified in the authentication statement of the assertion used to establish the ses-
1344 sion being terminated. If a <SessionIndex> element was present in the authentication statement, an
1345 identical <SessionIndex> **MUST** be present in the <LogoutRequest>. If no <SessionIndex> el-
1346 ement was present in the authentication statement, the <SessionIndex> **MUST** be omitted from the
1347 <LogoutRequest>.

1348 **RelayState** [Optional]

1349 This may contain state information that will be relayed back in the response. This data **SHOULD** be integrity-
1350 protected by the request author and **MAY** have other protections placed on it by the request author. An
1351 example of such protection is confidentiality.

1352 consent [Optional]
1353 Indicates whether or not consent has been obtained from a user in sending this message.

1354 The schema fragment is as follows:

```
1355 <xs:element name="LogoutRequest" type="LogoutRequestType"/>
1356 <xs:complexType name="LogoutRequestType">
1357   <xs:complexContent>
1358     <xs:extension base="saml:RequestAbstractType">
1359       <xs:sequence>
1360         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1361         <xs:element ref="ProviderID"/>
1362         <xs:element ref="saml:NameIdentifier"/>
1363         <xs:element name="SessionIndex" type="xs:string" minOccurs="0"/>
1364         <xs:element ref="RelayState" minOccurs="0"/>
1365       </xs:sequence>
1366       <xs:attribute ref="consent" use="optional"/>
1367     </xs:extension>
1368   </xs:complexContent>
1369 </xs:complexType>
1370
1371
```

1372 3.5.1.2. Example

```
1373
1374 <LogoutRequest RequestID="d4769303-7c33-4d65-931f-ddeb19fa6a73"
1375   MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
1376   IssueInstant="2001-12-17T09:30:47Z">
1377   <ds:Signature>...</ds:Signature>
1378   <ProviderID>http://ServiceProvider.com</ProviderID>
1379   <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
1380     Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be3
1381 5-cc9e7db39e2b</saml:NameIdentifier>
1382   <SessionIndex>3</SessionIndex>
1383   <RelayState>R0lGODlhcgGSA LMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1384 </LogoutRequest>
```

1385 3.5.2. Response

1386 The recipient MUST return a <LogoutResponse> message, which is of type StatusResponseType.

1387 This message SHOULD be signed.

1388 3.5.2.1. Element <LogoutResponse>

1389 The elements of the message are as follows:

1390 Extension [Optional]
1391 Optional container for protocol extensions established by agreement between providers.

1392 ProviderID [Required]
1393 The identifier of the provider responding.

1394 Status [Required]
1395 A status code that indicates the result of the request.

1396 RelayState [Optional]
1397 This contains state information that may have appeared in the request, and is being relayed back to the sender.

1398 The schema fragment is as follows:

```
1399 <xs:element name="LogoutResponse" type="StatusResponseType" />
1400
1401
```

1402 3.5.2.2. Example

```
1403
1404 <LogoutResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
1405 InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
1406 MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
1407 Recipient="http://ServiceProvider.com">
1408 <ds:Signature>...</ds:Signature>
1409 <ProviderID>http://IdentityProvider.com</ProviderID>
1410 <samlp:Status>
1411 <samlp:StatusCode Value="samlp:Success" />
1412 </samlp:Status>
1413 <RelayState>R01GOD1hcgGSALMAAAQCAEMmCZt uMFQxDS8b</RelayState>
1414 </LogoutResponse>
```

1415 3.5.2.3. Processing Rules

1416 If <RelayState> contains a value, the recipient **MUST** include this value in unmodified form in the <RelayState>
1417 element of the response.

1418 If the Principal's identity federation is between the identity provider and an affiliation group in which the service
1419 provider is a member, then the NameQualifier attribute **MUST** contain the unique identifier of the affiliation group.
1420 Otherwise, it **MUST** contain the unique identifier of the service provider. This attribute **MUST** be used by the providers
1421 to identify the specific identity federation of the Principal who is logging out.

1422 Other unique processing rules apply based on whether the message receiver is an identity provider or a service provider.

1423 3.5.2.3.1. Identity Provider Processing Rules

1424 When an identity provider receives the <LogoutRequest> message, the identity provider **MUST** validate that any
1425 signature present on the message is the signature of a service provider to which the identity provider provided an
1426 authentication assertion for the current session. If that holds, the identity provider **SHOULD** do the following:

1427 Send a <LogoutRequest> message to each service provider for which the identity provider provided authentication
1428 assertions in the current session, other than the originator of the <LogoutRequest>.

1429 Send a <LogoutRequest> message to the identity provider on behalf of whom the identity provider proxied the user's
1430 authentication, unless the second identity provider is the originator of the <LogoutRequest>.

1431 Terminate the Principal's current session as specified by the <saml:NameIdentifier> element.

1432 If an error occurs during this further processing of the logout (for example, relying service providers may not all
1433 implement the Single Logout profile used by the requesting service provider), then the identity provider **MUST** respond
1434 to the original requester with a <LogoutResponse> message, indicating the status of the logout request. The value
1435 "lib:UnsupportedProfile" is provided for a second-level <samlp:StatusCode>, indicating that a service provider
1436 should retry the <LogoutRequest> using a different profile.

1437 3.5.2.3.2. Service Provider Processing Rules

1438 When the service provider receives the <LogoutRequest> message, the service provider **MUST** validate the identity
1439 provider's signature contained in the <ds:Signature> element. If the signature is that of the identity provider that
1440 provided the authentication for the Principal's current session, the service provider **MUST** invalidate the Principal's
1441 session referred to in the <saml:NameIdentifier> element.

1442 3.6. Name Identifier Mapping Protocol

1443 When a service provider requires a name identifier for a Principal with which it has an identity federation relationship,
1444 but which references an identity federation between the identity provider and another service provider, it can use this
1445 protocol to obtain such an identifier. This allows the requesting provider to communicate with the other service
1446 provider about the Principal without an identity federation for the Principal between them. The resulting value
1447 SHOULD be encrypted so as to obscure the actual value from anyone but the second service provider. To the requester,
1448 it will be an opaque (and one-time) value.

1449 Upon receipt of a <NameIdentifierMappingRequest> message, an identity provider that supports this protocol
1450 MUST respond with a <NameIdentifierMappingResponse> message.

1451 **3.6.1. Request**

1452 The requesting service provider sends a <NameIdentifierMappingRequest> message to the identity provider
1453 which can provide the desired federated name identifier.

1454 The <NameIdentifierMappingRequest> message MUST be signed.

1455 **3.6.1.1. Element <NameIdentifierMappingRequest>**

1456 The elements and attributes are as follows:

1457 `Extension` [Optional]

1458 Optional container for protocol extensions established by agreement between providers.

1459 `ProviderID` [Required]

1460 The unique identifier of the provider that is sending this message.

1461 `saml:NameIdentifier` [Required]

1462 The name identifier of the Principal for whom the requester is obtaining a mapped identifier. See the
1463 processing rules below for additional information on the content of this element.

1464 `TargetNamespace` [Required]

1465 The unique identifier of the service provider or affiliation group for whom the requester needs the name
1466 identifier of the Principal to subsequently communicate with.

1467 `consent` [Optional]

1468 Indicates whether or not consent has been obtained from a user in sending this message.

1469 The schema fragment is as follows:

```
1470 <xs:element name="NameIdentifierMappingRequest" type="NameIdentifierMappingRequestType"/>
1471 <xs:complexType name="NameIdentifierMappingRequestType">
1472   <xs:complexContent>
1473     <xs:extension base="samlp:RequestAbstractType">
1474       <xs:sequence>
1475         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1476         <xs:element ref="ProviderID"/>
1477         <xs:element ref="saml:NameIdentifier"/>
1478         <xs:element name="TargetNamespace" type="md:entityIDType"/>
1479       </xs:sequence>
1480       <xs:attribute ref="consent" use="optional"/>
1481     </xs:extension>
1482   </xs:complexContent>
1483 </xs:complexType>
1484
1485
```

1486 **3.6.1.2. Example**

```

1487
1488     <NameIdentifierMappingRequest RequestID="e9c2-eb65-4bce-ab8f-4becdf229815"
1489       MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">
1490       <ds:Signature>...</ds:Signature>
1491       <ProviderID>http://RequestingServiceProvider.com</ProviderID>
1492       <saml:NameIdentifier NameQualifier="http://RequestingServiceProvider.com"
1493         Format="urn:liberty:iff:nameid:federated">e958019a</saml:NameIdentifier>
1494       <TargetNamespace>http://TargetServiceProvider.com</TargetNamespace>
1495     </NameIdentifierMappingRequest>

```

1496 3.6.2. Response

1497 The responding provider MUST return a <NameIdentifierMappingResponse> message.

1498 This message SHOULD be signed.

1499 3.6.2.1. Element <NameIdentifierMappingResponse>

1500 The elements of the message are as follows:

1501 Extension [Optional]

1502 Optional container for protocol extensions established by agreement between providers.

1503 ProviderID [Required]

1504 The identifier of the provider responding.

1505 Status [Required]

1506 A status code that indicates the reception and processing status of the message.

1507 saml:NameIdentifier [Optional]

1508 If the request is successful, contains the resulting mapped name identifier for the desired identity federation,
1509 which SHOULD be in encrypted form. See the processing rules below for additional information on the
1510 content of this element.

1511 The schema fragment is as follows:

```

1512 <xs:element name="NameIdentifierMappingResponse" type="NameIdentifierMappingResponseType" />
1513 <xs:complexType name="NameIdentifierMappingResponseType">
1514   <xs:complexContent>
1515     <xs:extension base="samlp:ResponseAbstractType">
1516       <xs:sequence>
1517         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1518         <xs:element ref="ProviderID" />
1519         <xs:element ref="samlp:Status" />
1520         <xs:element ref="saml:NameIdentifier" minOccurs="0" />
1521       </xs:sequence>
1522     </xs:extension>
1523   </xs:complexContent>
1524 </xs:complexType>
1525
1526

```

1527 3.6.2.2. Example

```

1528
1529     <NameIdentifierMappingResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
1530       InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
1531       MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
1532       Recipient="http://ServiceProvider.com">
1533       <ds:Signature>...</ds:Signature>
1534       <ProviderID>http://IdentityProvider.com</ProviderID>

```

```
1535         <samlp:Status>
1536             <samlp:StatusCode Value="samlp:Success" />
1537         </samlp:Status>
1538         <saml:NameIdentifier NameQualifier="http://TargetServiceProvider.com"
1539             Format="urn:liberty:iff:nameid:federated">gfhfrfe89u43</saml:Name
1540 Identifier>
1541     </NameIdentifierMappingResponse>
```

1542 3.6.3. Processing Rules

1543 The receiving provider **MUST** validate any signature present on the message. The signature on the message **MUST** be
1544 the signature of the <ProviderID> contained in the message. If the signature is not valid, the provider **MUST** ignore
1545 the message.

1546 In the request message's <saml:NameIdentifier>, if the Principal's identity federation is between the identity
1547 provider and an affiliation group in which the requesting service provider is a member, then the NameQualifier
1548 attribute **MUST** contain the unique identifier of the affiliation group. Otherwise, it **MUST** contain the unique identifier
1549 of the service provider.

1550 3.6.3.1. Recipient Processing Rules

1551 Supporting identity providers **MUST** respond with a <NameIdentifierMappingResponse> message.

1552 If the identity provider is unable to identify the Principal referenced in the request, then it **MUST** return a response
1553 message with a second-level <samlp:StatusCode> Value of *lib:UnknownPrincipal*.

1554 If the identity provider is unable to map the name identifier of the Principal to a federation between it
1555 and the <TargetNamespace> in the request, then it **MUST** return a response message with a second-level
1556 <samlp:StatusCode> Value of *lib:FederationDoesNotExist*.

1557 If the mapping is successful, then the response message's <saml:NameIdentifier> **MUST** contain a
1558 NameQualifier that matches the request's <TargetNamespace> element. It **SHOULD** be a one-time en-
1559 crypted value but **MAY** be the actual federated identifier value; the Format attribute **MUST** indicate this distinction.

1560 4. Schema Definition

```
1561 <?xml version="1.0" encoding="UTF-8"?>
1562 <xs:schema targetNamespace="urn:liberty:iff:2003-08"
1563   xmlns="urn:liberty:iff:2003-08"
1564   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1565   xmlns:ac="urn:liberty:ac:2003-08"
1566   xmlns:md="urn:liberty:metadata:2003-08"
1567   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1568   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
1569   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
1570   elementFormDefault="qualified" attributeFormDefault="unqualified">
1571
1572   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
1573     schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
1574   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
1575     schemaLocation="oasis-sstc-saml-schema-protocol-1.1.xsd"/>
1576   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
1577     schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd"/>
1578   <xs:import namespace="urn:liberty:ac:2003-08" schemaLocation="liberty-authentication-conte
1579 xt-v1.2.xsd"/>
1580   <xs:import namespace="urn:liberty:metadata:2003-08" schemaLocation="liberty-metadata-v1.0.xsd"/>
1581
1582
1583   <xs:include schemaLocation="liberty-idff-utility-v1.0.xsd"/>
1584
1585   <xs:annotation>
1586     <xs:documentation>
1587 The source code in this XSD file was excerpted verbatim from:
1588
1589 Liberty ID-FF Protocols & Schema Specification
1590 Version 1.2
1591 12th November 2003
1592
1593 Copyright (c) 2003 Liberty Alliance participants, see
1594 http://www.projectliberty.org/specs/idff_copyrights.html
1595
1596 </xs:documentation>
1597 </xs:annotation>
1598 <xs:element name="ProviderID" type="md:entityIDType"/>
1599 <xs:element name="AffiliationID" type="md:entityIDType"/>
1600
1601 <xs:element name="AuthnRequest" type="AuthnRequestType"/>
1602 <xs:complexType name="AuthnRequestType">
1603   <xs:complexContent>
1604     <xs:extension base="samlp:RequestAbstractType">
1605       <xs:sequence>
1606         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1607         <xs:element ref="ProviderID"/>
1608         <xs:element ref="AffiliationID" minOccurs="0"/>
1609         <xs:element ref="NameIDPolicy" minOccurs="0"/>
1610         <xs:element name="ForceAuthn" type="xs:boolean" minOccurs="0"/>
1611         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
1612         <xs:element ref="ProtocolProfile" minOccurs="0"/>
1613         <xs:element name="AssertionConsumerServiceID" type="xs:string" minOccurs="0"/>
1614         <xs:element ref="RequestAuthnContext" minOccurs="0"/>
1615         <xs:element ref="RelayState" minOccurs="0"/>
1616         <xs:element ref="Scoping" minOccurs="0"/>
1617       </xs:sequence>
1618       <xs:attribute ref="consent" use="optional"/>
1619     </xs:extension>
1620   </xs:complexContent>
1621 </xs:complexType>
1622 <xs:simpleType name="NameIDPolicyType">
1623   <xs:restriction base="xs:string">
1624     <xs:enumeration value="none"/>
1625     <xs:enumeration value="onetime"/>
```

```

1626         <xs:enumeration value="federated"/>
1627         <xs:enumeration value="any"/>
1628     </xs:restriction>
1629 </xs:simpleType>
1630 <xs:element name="NameIDPolicy" type="NameIDPolicyType"/>
1631 <xs:simpleType name="AuthnContextComparisonType">
1632     <xs:restriction base="xs:string">
1633         <xs:enumeration value="exact"/>
1634         <xs:enumeration value="minimum"/>
1635         <xs:enumeration value="better"/>
1636     </xs:restriction>
1637 </xs:simpleType>
1638 <xs:complexType name="ScopingType">
1639     <xs:sequence>
1640         <xs:element name="ProxyCount" type="xs:nonNegativeInteger" minOccurs="0"/>
1641         <xs:element ref="IDPList" minOccurs="0"/>
1642     </xs:sequence>
1643 </xs:complexType>
1644 <xs:element name="Scoping" type="ScopingType"/>
1645 <xs:element name="RelayState" type="xs:string"/>
1646 <xs:element name="ProtocolProfile" type="xs:anyURI"/>
1647 <xs:element name="RequestAuthnContext">
1648     <xs:complexType>
1649         <xs:sequence>
1650             <xs:choice>
1651                 <xs:element name="AuthnContextClassRef" type="xs:anyURI" maxOccurs="unbounded"/>
1652                 <xs:element name="AuthnContextStatementRef" type="xs:anyURI" maxOccurs="unbounded"/>
1653             </xs:choice>
1654             <xs:element name="AuthnContextComparison" type="AuthnContextComparisonType" m
1656 inOccurs="0"/>
1657         </xs:sequence>
1658     </xs:complexType>
1659 </xs:element>
1660
1661 <xs:element name="AuthnResponse" type="AuthnResponseType"/>
1662 <xs:complexType name="AuthnResponseType">
1663     <xs:complexContent>
1664         <xs:extension base="saml:ResponseType">
1665             <xs:sequence>
1666                 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1667                 <xs:element ref="ProviderID"/>
1668                 <xs:element ref="RelayState" minOccurs="0"/>
1669             </xs:sequence>
1670             <xs:attribute ref="consent" use="optional"/>
1671         </xs:extension>
1672     </xs:complexContent>
1673 </xs:complexType>
1674 <xs:element name="Assertion" type="AssertionType" substitutionGroup="saml:Assertion"/>
1675 <xs:complexType name="AssertionType">
1676     <xs:complexContent>
1677         <xs:extension base="saml:AssertionType">
1678             <xs:attribute name="InResponseTo" type="xs:NCName" use="optional"/>
1679         </xs:extension>
1680     </xs:complexContent>
1681 </xs:complexType>
1682 <xs:complexType name="SubjectType">
1683     <xs:complexContent>
1684         <xs:extension base="saml:SubjectType">
1685             <xs:sequence>
1686                 <xs:element ref="IDPProvidedNameIdentifier"/>
1687             </xs:sequence>
1688         </xs:extension>
1689     </xs:complexContent>
1690 </xs:complexType>
1691 <xs:element name="Subject" type="SubjectType" substitutionGroup="saml:Subject"/>
1692 <xs:element name="EncryptableNameIdentifier" type="EncryptableNameIdentifierType"

```

```

1693     substitutionGroup="saml:NameIdentifier"/>
1694 <xs:complexType name="EncryptableNameIdentifierType">
1695   <xs:simpleContent>
1696     <xs:extension base="saml:NameIdentifierType">
1697       <xs:attribute name="IssueInstant" type="xs:dateTime"/>
1698       <xs:attribute name="Nonce" type="xs:string"/>
1699     </xs:extension>
1700   </xs:simpleContent>
1701 </xs:complexType>
1702 <xs:element name="EncryptedNameIdentifier" type="EncryptedNameIdentifierType"/>
1703 <xs:complexType name="EncryptedNameIdentifierType">
1704   <xs:sequence>
1705     <xs:element ref="xenc:EncryptedData"/>
1706     <xs:element ref="xenc:EncryptedKey" minOccurs="0"/>
1707   </xs:sequence>
1708 </xs:complexType>
1709
1710 <xs:element name="AuthenticationStatement" type="AuthenticationStatementType" substitutionGroup="sa
1711 ml:Statement"/>
1712 <xs:complexType name="AuthenticationStatementType">
1713   <xs:complexContent>
1714     <xs:extension base="saml:AuthenticationStatementType">
1715       <xs:sequence>
1716         <xs:element ref="AuthnContext" minOccurs="0"/>
1717       </xs:sequence>
1718       <xs:attribute name="ReauthenticateOnOrAfter" type="xs:dateTime" use="optional"/>
1719       <xs:attribute name="SessionIndex" type="xs:string" use="optional"/>
1720     </xs:extension>
1721   </xs:complexContent>
1722 </xs:complexType>
1723 <xs:element name="AuthnContext">
1724   <xs:complexType>
1725     <xs:sequence>
1726       <xs:element name="AuthnContextClassRef" type="xs:anyURI" minOccurs="0"/>
1727       <xs:choice>
1728         <xs:element ref="ac:AuthenticationContextStatement"/>
1729         <xs:element name="AuthnContextStatementRef" type="xs:anyURI"/>
1730       </xs:choice>
1731     </xs:sequence>
1732   </xs:complexType>
1733 </xs:element>
1734 <xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType"/>
1735 <xs:complexType name="AuthnRequestEnvelopeType">
1736   <xs:complexContent>
1737     <xs:extension base="RequestEnvelopeType">
1738       <xs:sequence>
1739         <xs:element ref="AuthnRequest"/>
1740         <xs:element ref="ProviderID"/>
1741         <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
1742         <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
1743         <xs:element ref="IDPList" minOccurs="0"/>
1744         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
1745       </xs:sequence>
1746     </xs:extension>
1747   </xs:complexContent>
1748 </xs:complexType>
1749 <xs:complexType name="RequestEnvelopeType">
1750   <xs:sequence>
1751     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1752   </xs:sequence>
1753 </xs:complexType>
1754 <xs:element name="IDPList" type="IDPListType"/>
1755 <xs:complexType name="IDPListType">
1756   <xs:sequence>
1757     <xs:element ref="IDPEntries"/>
1758     <xs:element ref="GetComplete" minOccurs="0"/>
1759   </xs:sequence>

```

```

1760 </xs:complexType>
1761 <xs:element name="IDPEntry">
1762   <xs:complexType>
1763     <xs:sequence>
1764       <xs:element ref="ProviderID" />
1765       <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
1766       <xs:element name="Loc" type="xs:anyURI"/>
1767     </xs:sequence>
1768   </xs:complexType>
1769 </xs:element>
1770 <xs:element name="IDPEntries">
1771   <xs:complexType>
1772     <xs:sequence>
1773       <xs:element ref="IDPEntry" maxOccurs="unbounded"/>
1774     </xs:sequence>
1775   </xs:complexType>
1776 </xs:element>
1777 <xs:element name="GetComplete" type="xs:anyURI"/>
1778 <xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType"/>
1779 <xs:complexType name="AuthnResponseEnvelopeType">
1780   <xs:complexContent>
1781     <xs:extension base="ResponseEnvelopeType">
1782       <xs:sequence>
1783         <xs:element ref="AuthnResponse" />
1784         <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI" />
1785       </xs:sequence>
1786     </xs:extension>
1787   </xs:complexContent>
1788 </xs:complexType>
1789 <xs:complexType name="ResponseEnvelopeType">
1790   <xs:sequence>
1791     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1792   </xs:sequence>
1793 </xs:complexType>
1794 <xs:element name="RegisterNameIdentifierRequest" type="RegisterNameIdentifierRequestType"/>
1795 <xs:complexType name="RegisterNameIdentifierRequestType">
1796   <xs:complexContent>
1797     <xs:extension base="samlp:RequestAbstractType">
1798       <xs:sequence>
1799         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1800         <xs:element ref="ProviderID"/>
1801         <xs:element ref="IDPProvidedNameIdentifier"/>
1802         <xs:element ref="SPProvidedNameIdentifier"/>
1803         <xs:element ref="OldProvidedNameIdentifier"/>
1804         <xs:element ref="RelayState" minOccurs="0"/>
1805       </xs:sequence>
1806     </xs:extension>
1807   </xs:complexContent>
1808 </xs:complexType>
1809 <xs:element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1810 <xs:element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1811 <xs:element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1812 <xs:element name="RegisterNameIdentifierResponse" type="StatusResponseType"/>
1813 <xs:complexType name="StatusResponseType">
1814   <xs:complexContent>
1815     <xs:extension base="samlp:ResponseAbstractType">
1816       <xs:sequence>
1817         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1818         <xs:element ref="ProviderID"/>
1819         <xs:element ref="samlp:Status"/>
1820         <xs:element ref="RelayState" minOccurs="0"/>
1821       </xs:sequence>
1822     </xs:extension>
1823   </xs:complexContent>
1824 </xs:complexType>
1825 <xs:element name="FederationTerminationNotification" type="FederationTerminationNotificationType"/>
1826

```

```

1827 <xs:complexType name="FederationTerminationNotificationType">
1828 <xs:complexContent>
1829 <xs:extension base="samlp:RequestAbstractType">
1830 <xs:sequence>
1831 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1832 <xs:element ref="ProviderID"/>
1833 <xs:element ref="saml:NameIdentifier"/>
1834 </xs:sequence>
1835 <xs:attribute ref="consent" use="optional"/>
1836 </xs:extension>
1837 </xs:complexContent>
1838 </xs:complexType>
1839 <xs:element name="LogoutRequest" type="LogoutRequestType"/>
1840 <xs:complexType name="LogoutRequestType">
1841 <xs:complexContent>
1842 <xs:extension base="samlp:RequestAbstractType">
1843 <xs:sequence>
1844 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1845 <xs:element ref="ProviderID"/>
1846 <xs:element ref="saml:NameIdentifier"/>
1847 <xs:element name="SessionIndex" type="xs:string" minOccurs="0"/>
1848 <xs:element ref="RelayState" minOccurs="0"/>
1849 </xs:sequence>
1850 <xs:attribute ref="consent" use="optional"/>
1851 </xs:extension>
1852 </xs:complexContent>
1853 </xs:complexType>
1854 <xs:element name="LogoutResponse" type="StatusResponseType"/>
1855 <xs:element name="NameIdentifierMappingRequest" type="NameIdentifierMappingRequestType"/>
1856 <xs:complexType name="NameIdentifierMappingRequestType">
1857 <xs:complexContent>
1858 <xs:extension base="samlp:RequestAbstractType">
1859 <xs:sequence>
1860 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1861 <xs:element ref="ProviderID"/>
1862 <xs:element ref="saml:NameIdentifier"/>
1863 <xs:element name="TargetNamespace" type="md:entityIDType"/>
1864 </xs:sequence>
1865 <xs:attribute ref="consent" use="optional"/>
1866 </xs:extension>
1867 </xs:complexContent>
1868 </xs:complexType>
1869 <xs:element name="NameIdentifierMappingResponse" type="NameIdentifierMappingResponseType"/>
1870 <xs:complexType name="NameIdentifierMappingResponseType">
1871 <xs:complexContent>
1872 <xs:extension base="samlp:ResponseAbstractType">
1873 <xs:sequence>
1874 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1875 <xs:element ref="ProviderID"/>
1876 <xs:element ref="samlp:Status"/>
1877 <xs:element ref="saml:NameIdentifier" minOccurs="0"/>
1878 </xs:sequence>
1879 </xs:extension>
1880 </xs:complexContent>
1881 </xs:complexType>
1882 </xs:schema>
1883
1884

```

References

1886 Normative

- 1887 [LibertyBindProf] Cantor, Scott, Kemp, John , eds. "Liberty ID-FF Bindings and Profiles Specification," Version 1.2,
1888 Liberty Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 1889 [LibertyGlossary] Wason, Thomas, eds. "Liberty Technical Glossary," Version 1.2, Liberty Alliance Project (12
1890 November 2003). <http://www.projectliberty.org/specs>
- 1891 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.0, Liberty
1892 Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 1893 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1894 Engineering Task Force (March 1997). <ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt>
- 1895 [RFC3280] Housley, R., eds. (April 2002). "Internet X.509 Public Key Infrastructure Certificate and Certifi-
1896 cate Revocation List (CRL) Profile," RFC 3280, The Internet Engineering Task Force <http://www.rfc->
1897 [editor.org/rfc/rfc3280.txt](http://www.rfc-editor.org/rfc/rfc3280.txt)
- 1898 [SAMLCore11] Maler, E., Mishra, P., Philpott, R., eds. (27 May 2003). "Assertions and Protocol for the
1899 OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, ver-
1900 sion 1.1, Organization for the Advancement of Structured Information Standards <http://www.oasis->
1901 [open.org/committees/documents.php?wg_abbrev=security](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)
- 1902 [SAMLBind11] Maler, E., Mishra, P., Philpott, R., eds. (27 May 2003). "Bindings and Profiles for the
1903 OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, ver-
1904 sion 1.1, Organization for the Advancement of Structured Information Standards <http://www.oasis->
1905 [open.org/committees/documents.php?wg_abbrev=security](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)
- 1906 [Schema1] Thompson, H.S., Beech, D., Maloney, M., Mendleson, N., eds. (May 2002). "XML Schema Part 1:
1907 Structures," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlschema-1/>
- 1908 [Schema2] Biron, P.V., Malhotra, A., eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation, World
1909 Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>
- 1910 [XMLDsig] Eastlake, D., Reagle, J., Solo, D., eds. (12 Feb 20002). "XML-Signature Syntax and Processing,"
1911 Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 1912 [XMLCanon] Boyer, J., Eastlake, D., Reagle, J., eds. (18 July 2002). "Exclusive XML Canonicalization,"
1913 Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xml-exc-c14n>
- 1914 [xmlesc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing,"
1915 W3C Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlesc-core/>

1916 Informative

- 1917 [LibertyImplGuide] Kemp, John, eds. "Liberty ID-FF Implementation Guidelines," Version 1.2, Liberty Alliance
1918 Project (.) <http://www.projectliberty.org/specs>