



Liberty ID-FF Protocols and Schema Specification

Version: 1.2-errata-v3.0

Editors:

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

Contributors:

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

Abstract:

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

Filename: draft-liberty-idff-protocols-schema-1.2-errata-v3.0.pdf

1

Notice

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

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

15 Copyright © 2004-2005 ADAE; Adobe Systems; America Online, Inc.; American Express Company; Avatier
16 Corporation; Axalto; Bank of America Corporation; BIPAC; Computer Associates International, Inc.; DataPower
17 Technology, Inc.; Diversinet Corp.; Enosis Group LLC; Entrust, Inc.; Epok, Inc.; Ericsson; Fidelity Investments;
18 Forum Systems, Inc. ; France Telecom; Gamefederation; Gemplus; General Motors; Giesecke & Devrient GmbH;
19 Hewlett-Packard Company; IBM Corporation; Intel Corporation; Intuit Inc.; Kantega; Kayak Interactive; MasterCard
20 International; Mobile Telephone Networks (Pty) Ltd; NEC Corporation; Netegrity, Inc.; NeuStar, Inc.; Nippon
21 Telegraph and Telephone Corporation; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OpenNetwork; Oracle
22 Corporation; Ping Identity Corporation; Royal Mail Group plc; RSA Security Inc.; SAP AG; Senforce; Sharp
23 Laboratories of America; Sigaba; SmartTrust; Sony Corporation; Sun Microsystems, Inc.; Telefonica Moviles, S.A.;
24 Trusted Network Technologies.; Trustgenix; UTI; VeriSign, Inc.; Vodafone Group Plc. All rights reserved.

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

31 **Contents**

32 [1. Introduction](#) 4
33 [2. Schema Declarations](#) 6
34 [3. Protocols](#) 7
35 [4. Schema Definition](#) 43
36 [References](#) 48

37 1. Introduction

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

41 1.1. Notation

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

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

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

53 Listings of schemas appear like this.

54 Listings of instance fragments appear like this.

55 The following namespaces are referred to in this document:

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

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

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

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

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

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

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

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

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

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

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

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

72 1.2. Overview

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

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

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

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

87 2. Schema Declarations

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

90 2.1. Schema Header and Namespace Declarations

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

```
92 <?xml version="1.0" encoding="UTF-8"?>
93 <xs:schema targetNamespace="urn:liberty:iff:2003-08"
94   xmlns="urn:liberty:iff:2003-08"
95   xmlns:xs="http://www.w3.org/2001/XMLSchema"
96   xmlns:ac="urn:liberty:ac:2003-08"
97   xmlns:md="urn:liberty:metadata:2003-08"
98   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
99   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
100  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
101  elementFormDefault="qualified" attributeFormDefault="unqualified">
102
103   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
104     schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
105   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
106     schemaLocation="oasis-sstc-saml-schema-protocol-1.1.xsd"/>
107   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
108     schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
109   <xs:import namespace="urn:liberty:ac:2003-
110 08" schemaLocation="liberty-authentication-context-v1.3.xsd"/>
111   <xs:import namespace="urn:liberty:metadata:2003-0
112 8" schemaLocation="liberty-metadata-v1.1.xsd"/>
113
114   <xs:include schemaLocation="liberty-idff-utility-v1.0.xsd"/>
115
116   <xs:annotation>
117     <xs:documentation> ### IMPORTANT NOTICE ###
118
119     The source code in this XSD file was excerpted verbatim from:
120
121     Liberty Protocols and Schema Specification
122     Version 1.2-errata-v3.0
123     19 May 2005
124     Updated for WSF 1.1 release
125
126     Copyright (c) 2003-2005 Liberty Alliance participants, see
127     http://www.projectliberty.org/specs/idff_copyrights.html
128
129   </xs:documentation>
130 </xs:annotation>
131
132
```

133 2.1.1. Type and Element Declarations

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

```
135   <xs:element name="ProviderID" type="md:entityIDType"/>
136   <xs:element name="AffiliationID" type="md:entityIDType"/>
137
138
139
```

140 **3. Protocols**

141 The Liberty protocol suite consists of the following protocols:

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

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

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

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

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

150 **3.1. General Requirements**

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

152 **3.1.1. XML Signature**

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

155 **3.1.2. Protocol and Assertion Versioning**

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

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

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

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

- 169 • If the XML element containing the `MajorVersion` and `MinorVersion` attributes is in the Liberty ID-FF
170 namespace (`urn:liberty:iff:2003-08`) or has an `xsi:type` attribute value in this namespace, then the
171 `MajorVersion` **MUST** be 1 and the `MinorVersion` **MUST** be 2.
- 172 • If the element or its type is in a SAML 1.1 namespace (`urn:oasis:names:tc:SAML:1.0:assertion` or
173 `urn:oasis:names:tc:SAML:1.0:protocol`), then the values **MUST** be 1 and 1 respectively.

174 3.1.3. Provider and Affiliation ID Uniqueness

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

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

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

182 3.1.4. Name Identifier Construction

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

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

192 3.1.5. Signature Verification

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

199 3.1.6. Security

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

205 3.1.7. Time Values

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

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

212 **3.1.8. Time Synchronization**

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

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

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

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

221 • `<LogoutRequest>` is received

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

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

225 **3.1.9. Response Status Codes**

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

230 **3.1.10. Use of `<Extension>` in Protocols**

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

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

239 **3.1.11. Interoperation with Previous Liberty Implementations**

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

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

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

- 248 • When in doubt, 1.2 implementations SHOULD send 1.1 requests and notifications when the older specification
249 meets the requirements of the transaction.
- 250 • Version 1.2 implementations MUST respond to older requests with responses matching the version of the request.
- 251 • These rules apply to all of the request/response protocols and asynchronous notifications defined in this specifica-
252 tion.

253 3.1.11.1. Deprecation of ID-FF 1.1 Name Identifier Practices

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

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

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

271 3.1.12. Use of the consent attribute

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

274 The following values are defined for this attribute:

- 275 • *urn:liberty:consent:obtained* indicates that a user's consent has been obtained by the sender of the message. If the
276 message sender uses this value, they SHOULD sign the message such that the signature covers this attribute.
- 277 • *urn:liberty:consent:obtained:prior* indicates that a user's consent has been obtained by the sender of the message
278 at some point prior to this action. If the message sender uses this value, they SHOULD sign the message such that
279 the signature covers this attribute.
- 280 • *urn:liberty:consent:obtained:current:implicit* indicates that a user's consent has been implicitly obtained by the
281 sender of the message during this action as part of a broader indication of consent. If the message sender uses this
282 value, they SHOULD sign the message such that the signature covers this attribute. Implicit consent is typically
283 more proximal to the action in time and presentation than prior consent, such as part of a session of activities.
- 284 • *urn:liberty:consent:obtained:current:explicit* indicates that a user's consent has been explicitly obtained by the
285 sender of the message during this action. If the message sender uses this value, they SHOULD sign the message
286 such that the signature covers this attribute.

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

290 **3.2. Single Sign-On and Federation Protocol**

291 The Single Sign-On and Federation Protocol defines a request and response protocol by which single sign-on and

292 identity federation occurs. The protocol is conducted between a service provider and one or more identity providers.

293 The protocol works as follows (note that step 1 is optional):

294 1. A service provider issues an `<AuthnRequest>` to an identity provider, instructing the identity provider to provide

295 an authentication assertion to the service provider. Optionally, the service provider MAY request that the identity be

296 federated.

297 2. The identity provider responds with either an `<AuthnResponse>` containing authentication assertions to the service

298 provider or an artifact that can be de-referenced into an authentication assertion. Additionally, the identity provider

299 potentially federates the Principal's identity at the identity provider with the Principal's identity at the service provider.

300 **Note:**

301 Under certain conditions, an identity provider may unilaterally (without receiving an authentication request)

302 issue an authentication response to a service provider.

303 The identity provider may be proxying for an *authenticating identity provider*, in which case, this protocol may be

304 repeated between the recipient of the original `<AuthnRequest>`, and other identity providers (see [Section 3.2.2.7](#)).

305 **3.2.1. Request**

306 The service provider issues an initial `<AuthnRequest>` to an identity provider. A set of parameters is included in the

307 request that allows the service provider to specify desired behavior at identity providers in processing the request. A

308 requester can control the following identity provider behaviors:

- 309 • Prompt the Principal for credentials if the Principal is not presently authenticated.
- 310 • Prompt the Principal for credentials, even if the Principal is presently authenticated.
- 311 • Federate the Principal's identity at the identity provider with the Principal's identity at the requester.
- 312 • Issue an anonymous and temporary identifier for the Principal to the service provider.
- 313 • Use a specific protocol profile in responding to the request.
- 314 • Use a specific authentication context (for example, smartcard-based authentication vs. username/password-based
- 315 authentication).
- 316 • Restrict the ability of the recipient to proxy the authentication request to additional identity providers.

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

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

321 3.2.1.1. Element <AuthnRequest>

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

325 The elements of the request are as follows:

326 `Extension` [Optional]

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

330 `ProviderID` [Required]

331 The requester's unique identifier.

332 `AffiliationID` [Optional]

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

334 `NameIDPolicy` [Optional]

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

336 `IsPassive` [Optional]

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

341 `ForceAuthn` [Optional]

342 Controls whether the identity provider authenticates the Principal regardless of whether the Principal is
343 already authenticated. This element is specified only when <IsPassive> is "false." If <ForceAuthn>
344 is "true," specifies that the identity provider MUST always authenticate the Principal, regardless of whether
345 the Principal is presently authenticated. If "false," specifies that the identity provider MUST re-authenticate
346 the user only if the Principal is not presently authenticated. If not specified, "false" is presumed.

347 `ProtocolProfile` [Optional]

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

350 `AssertionConsumerServiceID` [Optional]

351 Used to direct the identity provider to use a specific assertion consumer service URL at the service provider.
352 It references an element in the provider's metadata with a matching id attribute.

353 `RequestAuthnContext` [Optional]

354 Information describing which authentication context the requester desires the identity provider to use in
355 authenticating the Principal.

356 `RelayState` [Optional]

357 This contains state information that will be relayed back in the response. This data SHOULD be integrity-
358 protected by the request author and MAY have other protections placed on it by the request author. An

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

362 `Scoping` [Optional]

363 Specifies any preferences on the number and specific identifiers of additional identity providers through which
364 the authentication request may be proxied. See [Section 3.2.2.7](#) for rules regarding the proxying of identity
365 providers.

366 The requester may also choose not to include this element, in which case, the recipient of the message MAY
367 act as a proxy, according to the rules in [Section 3.2.2.7](#)

368 `consent` [Optional]

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

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

372 `AuthnContextClassRef` [Optional]

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

375 `AuthnContextStatementRef` [Optional]

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

378 `AuthnContextComparison` [Optional]

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

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

385 `ProxyCount` [Optional]

386 The upper limit on the number of proxying steps the requester wishes to specify for this authentication
387 request.

388 `IDPList` [Optional]

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

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

```
392 <xs:element name="AuthnRequest" type="AuthnRequestType"/>
393 <xs:complexType name="AuthnRequestType">
394   <xs:complexContent>
395     <xs:extension base="samlp:RequestAbstractType">
396       <xs:sequence>
397         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
398         <xs:element ref="ProviderID"/>
399         <xs:element ref="AffiliationID" minOccurs="0"/>
400         <xs:element ref="NameIDPolicy" minOccurs="0"/>
401         <xs:element name="ForceAuthn" type="xs:boolean" minOccurs="0"/>
402         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0"/>
403         <xs:element ref="ProtocolProfile" minOccurs="0"/>
404         <xs:element name="AssertionConsumerServiceID" type="xs:string" minOccurs="0"/>

```

```

405         <xs:element ref="RequestAuthnContext" minOccurs="0"/>
406         <xs:element ref="RelayState" minOccurs="0"/>
407         <xs:element ref="Scoping" minOccurs="0"/>
408     </xs:sequence>
409     <xs:attribute ref="consent" use="optional"/>
410 </xs:extension>
411 </xs:complexContent>
412 </xs:complexType>
413 <xs:simpleType name="NameIDPolicyType">
414     <xs:restriction base="xs:string">
415         <xs:enumeration value="none"/>
416         <xs:enumeration value="onetime"/>
417         <xs:enumeration value="federated"/>
418         <xs:enumeration value="any"/>
419     </xs:restriction>
420 </xs:simpleType>
421 <xs:element name="NameIDPolicy" type="NameIDPolicyType"/>
422 <xs:simpleType name="AuthnContextComparisonType">
423     <xs:restriction base="xs:string">
424         <xs:enumeration value="exact"/>
425         <xs:enumeration value="minimum"/>
426         <xs:enumeration value="maximum"/>
427         <xs:enumeration value="better"/>
428     </xs:restriction>
429 </xs:simpleType>
430 <xs:complexType name="ScopingType">
431     <xs:sequence>
432         <xs:element name="ProxyCount" type="xs:nonNegativeInteger" minOccurs="0"/>
433         <xs:element ref="IDPList" minOccurs="0"/>
434     </xs:sequence>
435 </xs:complexType>
436 <xs:element name="Scoping" type="ScopingType"/>
437 <xs:element name="RelayState" type="xs:string"/>
438 <xs:element name="ProtocolProfile" type="xs:anyURI"/>
439 <xs:element name="RequestAuthnContext">
440     <xs:complexType>
441         <xs:sequence>
442             <xs:choice>
443                 <xs:element name="AuthnContextClassRef" type="xs:anyURI" maxOccurs="unbounded"/>
444                 <xs:element name="AuthnContextStatementRef" type="xs:anyURI" maxOccurs="unbounded"/>
445             </xs:choice>
446             <xs:element name="AuthnContextComparison" type="AuthnContextComparisonType" ↵
447 minOccurs="0"/>
448         </xs:sequence>
449     </xs:complexType>
450 </xs:element>
451
452
453

```

454 3.2.1.2. Example

```

455
456     <lib:AuthnRequest RequestID="RPCUk211+GVz+t11LURp51oFvJXk"
457 MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
458 IssueInstant="2001-12-17T21:42:4Z" xmlns:lib="urn:liberty:iff:2003-08">
459     <ds:Signature> ... </ds:Signature>
460     <lib:ProviderID>http://ServiceProvider.com</lib:ProviderID>
461     <lib:NameIDPolicy>federate</lib:NameIDPolicy>
462     <lib:ForceAuthn>false</lib:ForceAuthn>
463     <lib:IsPassive>false</lib:IsPassive>
464     <lib:ProtocolProfile>http://projectliberty.org/profiles/b
465 rws-post</lib:ProtocolProfile>
466     <lib:RequestAuthnContext>
467     <lib:AuthnContextClassRef>http://projectliberty.org/schemas/
468 authctx/classes/Password-ProtectedTransport</lib:AuthnContextClassRef>
469     <lib:AuthnContextComparison>exact</lib:AuthnContextComparison>

```

```
470         </lib:RequestAuthnContext>
471         <lib:RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</lib:RelayState>
472         <lib:Scoping>
473         <lib:ProxyCount>1</lib:ProxyCount>
474     </lib:Scoping>
475 </lib:AuthnRequest>
```

476 3.2.2. Response

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

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

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

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

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

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

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

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

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

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

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

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

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

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

518 3.2.2.1. Element <AuthnResponse>

519 The type AuthnResponseType is extended from samlp:ResponseType.

520 The response contains the following elements:

521 Extension [Optional]

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

523 ProviderID [Required]

524 The identity provider's unique identifier.

525 RelayState [Optional]

526 This contains state information being relayed.

527 consent [Optional]

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

529 The schema fragment is as follows:

```
530 <xs:element name="AuthnResponse" type="AuthnResponseType" />
531 <xs:complexType name="AuthnResponseType">
532   <xs:complexContent>
533     <xs:extension base="samlp:ResponseType">
534       <xs:sequence>
535         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
536         <xs:element ref="ProviderID" />
537         <xs:element ref="RelayState" minOccurs="0" />
538       </xs:sequence>
539       <xs:attribute ref="consent" use="optional" />
540     </xs:extension>
541   </xs:complexContent>
542 </xs:complexType>
543
544
```

545 3.2.2.2. Element <AssertionType>

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

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

554 The schema fragment is as follows:

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



```
557     <xs:complexContent>
558         <xs:extension base="saml:AssertionType">
559             <xs:attribute name="InResponseTo" type="xs:NCName" use="optional"/>
560         </xs:extension>
561     </xs:complexContent>
562 </xs:complexType>
563
564
```

565 3.2.2.3. SubjectType and Related Types

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

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

571 **Format** [Required Except for Pre-1.2 Federations]

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

578 *urn:liberty:iff:nameid:federated*

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

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

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

583 *urn:liberty:iff:nameid:encrypted*

584 Used for identifiers that have been encrypted for use only by a specific provider. Such an
585 identifier **MUST** be a base-64 encoded <EncryptedNameIdentifier> element, defined
586 below. See the Name Identifier Encryption Profile in [\[LibertyBindProf\]](#) for more
587 information.

588 *urn:liberty:iff:nameid:entityID*

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

592 **NameQualifier** [Optional]

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

597 An issuer **MAY** omit this attribute if the containing element is in the subject of a statement in an assertion
598 that contains a single <saml:Audience> equal to the unique identifier of the value that this attribute would
599 have had. This typically means that an assertion issued during SSO to a service provider that is not acting as
600 part of an affiliation group can omit this attribute.

601 The above interpretation of this attribute applies **ONLY** if the **Format** attribute contains one of the four values
602 listed above. Otherwise, the identifier **MUST** be assumed to represent a pre-1.2 identity federation and **MUST**
603 be treated only as literal data.

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

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

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

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

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

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

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

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

635 The schema fragment is as follows:

```
636 <xs:complexType name="SubjectType">
637   <xs:complexContent>
638     <xs:extension base="saml:SubjectType">
639       <xs:sequence>
640         <xs:element ref="IDPProvidedNameIdentifier" minOccurs="0"/>
641       </xs:sequence>
642     </xs:extension>
643   </xs:complexContent>
644 </xs:complexType>
645 <xs:element name="Subject" type="SubjectType" substitutionGroup="saml:Subject"/>
646 <xs:element name="EncryptableNameIdentifier" type="EncryptableNameIdentifierType"
647   substitutionGroup="saml:NameIdentifier"/>
648 <xs:complexType name="EncryptableNameIdentifierType">
649   <xs:simpleContent>
650     <xs:extension base="saml:NameIdentifierType">
651       <xs:attribute name="IssueInstant" type="xs:dateTime"/>
652       <xs:attribute name="Nonce" type="xs:string" use="optional"/>
653     </xs:extension>
654   </xs:simpleContent>
655 </xs:complexType>
656 <xs:element name="EncryptedNameIdentifier" type="EncryptedNameIdentifierType"/>
```

```

657 <xs:complexType name="EncryptedNameIdentifierType">
658 <xs:sequence>
659 <xs:element ref="xenc:EncryptedData" />
660 <xs:element ref="xenc:EncryptedKey" minOccurs="0" />
661 </xs:sequence>
662 </xs:complexType>
663
664
665

```

666 3.2.2.4. Type AuthenticationStatementType

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

669 AuthnContext [Optional]

670 The context used by the identity provider in the authentication event that yielded this statement. Contains
671 either an authentication context statement or a reference to an authentication context statement. Optionally
672 contains a reference to an authentication context class.

673 ReauthenticateOnOrAfter [Optional]

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

676 SessionIndex [Required]

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

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

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

685 The schema fragment is as follows:

```

686 <xs:element name="AuthenticationStatement" type="AuthenticationStatementType" ↵
687 substitutionGroup="saml:Statement" />
688 <xs:complexType name="AuthenticationStatementType">
689 <xs:complexContent>
690 <xs:extension base="saml:AuthenticationStatementType">
691 <xs:sequence>
692 <xs:element name="AuthnContext" minOccurs="0">
693 <xs:complexType>
694 <xs:sequence>
695 <xs:element name="AuthnContextClassRef" type="xs:anyURI" minOccurs="0" />
696 <xs:choice>
697 <xs:element ref="ac:AuthenticationContextStatement" />
698 <xs:element name="AuthnContextStatementRef" type="xs:anyURI" />
699 </xs:choice>
700 </xs:sequence>
701 </xs:complexType>
702 </xs:element>
703 </xs:sequence>
704 <xs:attribute name="ReauthenticateOnOrAfter" type="xs:dateTime" use="optional" />
705 <xs:attribute name="SessionIndex" type="xs:string" use="required" />
706 </xs:extension>
707 </xs:complexContent>
708 </xs:complexType>

```

709
710

711 3.2.2.5. Example

```

712
713 <lib:AuthnResponse ResponseID="hhuujalbc744hGJn5Q9A5yvEIgS"
714   InResponseTo="Zon3WjJ2KL7j+bJu7MuIr 4Pt2go5" MajorVersion="1" MinorVersion="2"
715   consent="urn:liberty:consent:obtained" IssueInstant="2002-10-31T21:55:41Z">
716   <samlp:Status>
717     <samlp:StatusCode Value="samlp:Success"/>
718   </samlp:Status>
719   <lib:Assertion MajorVersion="1" MinorVersion="2"
720     AssertionID="e06e5a28-bc80-4ba6-9ecb-712949db686e"
721     Issuer="http://IdentityProvider.com" IssueInstant="2001-12-17T09:30:47Z"
722     InResponseTo="4e7c3772-4fa4-4a0f-99e8-7d719ff6067c">
723     <saml:Conditions NotBefore="2001-12-17T09:30:47Z"
724     NotOnOrAfter="2001-12-17T09:35:47Z">
725       <saml:AudienceRestrictionCondition>
726         <saml:Audience>http://ServiceProvider.com</saml:Audience>
727       </saml:AudienceRestrictionCondition>
728     </saml:Conditions>
729     <lib:AuthenticationStatement AuthenticationInstant="2001-12-17T09:30:47Z"
730       SessionIndex="3" ReauthenticateOnOrAfter="2001-12-17T11:30:47Z"
731       AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
732       <lib:Subject>
733         <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
734           Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-
735 4c68-be35-cc9e7db39e2b</saml:NameIdentifier>
736         <saml:SubjectConfirmation>
737           <saml:ConfirmationMethod>urn:oasis:names:tc:SAM
738 L:1.0:cm:bearer</saml:ConfirmationMethod>
739         </saml:SubjectConfirmation>
740         <IDPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
741           Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-
742 4c68-be35-cc9e7db39e2b</IDPProvidedNameIdentifier>
743         </lib:Subject>
744       </lib:AuthenticationStatement>
745     <ds:Signature>...</ds:Signature>
746   </lib:Assertion>
747   <lib:ProviderID>http://IdentityProvider.com</lib:ProviderID>
748   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
749 </lib:AuthnResponse>

```

750 3.2.2.6. Processing Rules

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

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

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

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

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

- 761 Optionally, a `<RelayState>` with a value understood by mutual agreement between the identity provider and service
762 provider so that the service provider knows how to handle subsequent interactions with the Principal after SSO. This
763 MAY be the URL of a resource at the service provider.
- 764 When an identity provider receives an authentication request, it MUST process the request according to the following
765 rules:
- 766 The `<ProviderID>` in the request MAY be the ProviderID of a known provider with which the identity provider has
767 established a relationship, or out of band means MAY be used to establish such a relationship.
- 768 The `<ProviderID>` MUST be resolvable to at least one (default) assertion consumer service URL at the requesting
769 provider that the identity provider may use when returning the corresponding assertion reference.
- 770 The following rules apply to choosing the appropriate URL to use:
- 771 If the `<AssertionConsumerServiceID>` element is provided, then the identity provider MUST search for the
772 value among the `id` attributes in the `<AssertionConsumerServiceURL>` elements in the provider's meta-
773 data to determine the URL to use. If no match can be found, then the provider MUST return an error
774 with a second-level `<samlp:StatusCode>` of `lib:InvalidAssertionConsumerServiceIndex` to the default URL (the
775 `<AssertionConsumerServiceURL>` with an `isDefault` attribute of "true").
- 776 If the `<AssertionConsumerServiceID>` element is not provided, then the identity provider MUST use the default
777 URL (the `<AssertionConsumerServiceURL>` with an `isDefault` attribute of "true").
- 778 `<ds:Signature>`, if present, MUST be the signature of the service provider as specified by the `<ProviderID>`.
- 779 If the requesting provider's `<AuthnRequestsSigned>` metadata element is "true", then any request messages it
780 generates MUST be signed. If an unsigned request is received, then the provider MUST return an error with a second-
781 level `<samlp:StatusCode>` of `lib:UnsignedAuthnRequest`.
- 782 If `<IsPassive>` is "true," the identity provider MUST NOT interact with the Principal and MUST NOT take control
783 of the user interface (if applicable).
- 784 The identity provider MUST attempt to authenticate the Principal if `<ForceAuthn>` is "true," regardless of whether
785 the Principal is presently authenticated, unless `<IsPassive>` is "true."
- 786 Success in authenticating the Principal is indicated by a status code of `samlp:Success` and a signed assertion containing
787 at least one statement of type `lib:AuthenticationStatementType` representing the Principal's authentication
788 information. Other types of statements may also be included, as defined by providers and other specifications.
- 789 Failure to authenticate the Principal is indicated by a status code other than `samlp:Success`. For failures, assertions
790 MUST NOT be included in the `<AuthnResponse>`.
- 791 `<AffiliationID>`, if present, MUST be the unique identifier of a known affiliation group with which the identity
792 provider has an established relationship, and of which the requesting provider is a member. If present, identity
793 providers MUST establish and resolve federations based on the specified affiliation, not the requesting provider. In
794 addition, identity providers MAY retrieve information regarding the other members of the affiliation group by querying
795 metadata (see [LibertyMetadata](#)) and present a list of members of the affiliation group to the Principal.
- 796 The following rules apply to the selection of name identifiers and the federation process:
- 797 If the `<NameIDPolicy>` element is omitted or "none", then the identity provider MUST return the name identifier(s)
798 corresponding to the federation that exists between the identity provider and the requesting provider or affiliation
799 group for the Principal. If no such federation exists, then an error with a second-level `<samlp:StatusCode>` of
800 `lib:FederationDoesNotExist` MUST be returned to the provider.

- 801 If the `<NameIDPolicy>` element is "onetime", then the `<saml:NameIdentifier>` element in the `<saml:Subject>`
802 element MUST be a temporary, one-time-use identifier for the Principal, with a `Format` attribute of
803 `urn:liberty:iff:nameid:one-time`.
- 804 If `<NameIDPolicy>` is *federated*, and if the Principal consents, then the identity provider MAY federate the Principal's
805 identity with the requesting provider (or the affiliation group if `<AffiliationID>` is present). If the identity provider
806 already has a previous federation on record for the Principal's identity at the requesting provider or affiliation group
807 (such as when a provider previously issued a `<FederationTerminationNotification>` which was not received
808 by the identity provider), then the identity provider SHOULD treat the request as if `<NameIDPolicy>` were *none*.
- 809 If `<NameIDPolicy>` is *any*, then the rules above for the values of *federated* and *onetime* MUST be followed, in that
810 order. Thus, a new federation may be created, an existing federation used, or a temporary identifier generated.
- 811 When including a Principal's federated identity in the response, the `<Subject>` element MUST include a
812 `<saml:NameIdentifier>`. If the service provider or affiliation group has set its own identifier for the Principal,
813 then the `<saml:NameIdentifier>` MUST be set to that value. The identifier set by the identity provider MUST
814 then be included in the `<IDPProvidedNameIdentifier>`.
- 815 If a one-time identifier is being used, or if the service provider or affiliation group has not set its own identifier for
816 the Principal, then the `<saml:NameIdentifier>` MUST contain the identifier set by the identity provider for the
817 Principal. The `<IDPProvidedNameIdentifier>` SHOULD be omitted in this case, since it would contain identical
818 information.
- 819 The `<Subject>` MUST contain a `<SubjectConfirmation>` element in accordance with the SSO profile used to
820 return the response (see [SAMLBind11] and [LibertyBindProf]).
- 821 When federating or in the case of a temporary value, the identity provider MUST adhere to the following rules in
822 generating the name identifier:
- 823 The name identifier MUST be unique across all Principals in the scope of that requester-identity provider relationship.
- 824 The name identifier for the specific Principal MUST be unique across all providers with which an identity federation
825 exists with the identity provider.
- 826 The identity provider MUST respond using the specified `<ProtocolProfile>`.
- 827 If `<RelayState>` contains a value, the identity provider MUST include this value in unmodified form in the
828 `<RelayState>` element of the returned authentication assertion.
- 829 The *InResponseTo* attribute in all generated `<Assertion>` elements in the `<AuthnResponse>` element MUST be set
830 to the value of the `RequestID` attribute in the corresponding `<AuthnRequest>` element. If there is no such request
831 because the identity provider is initiating the response on its own, then the attribute MUST NOT be included.
- 832 Additionally, if the `<RequestAuthnContext>` element is specified, the identity provider MUST authenticate the
833 Principal according to the following rules:
- 834 If one or more `<AuthnContextClassRef>` or `<AuthnContextStatementRef>` elements are included, then the
835 resulting authentication statement in the assertion (if any) MUST contain an authentication statement that conforms
836 to the class or statement specified. Additionally, the set of supplied elements MUST be evaluated as an ordered set,
837 where the first element is the most preferred authentication context class or statement. If none of the specified classes
838 or statements can be satisfied, the identity provider MUST not include an authentication statement in the resulting
839 assertion.
- 840 Additionally, if an `<AuthnContextComparison>` element is supplied, and one or more `<AuthnContextStatementRef>`
841 or `<AuthnContextClassRef>` elements are included, then the resulting authentication statement in the assertion (if
842 any) MUST follow the rule specified in the `<AuthnContextComparison>` element. If this requirement cannot be
843 satisfied, the identity provider MUST NOT include an authentication statement in the resulting assertion.

844 If `<AuthnContextComparison>` is specified and set to *exact*, then the resulting authentication statement in the
845 assertion (if any) MUST be the exact match of at least one of the authentication contexts specified.

846 If `<AuthnContextComparison>` is specified and set to *minimum*, then the resulting authentication statement in the
847 assertion (if any) MUST be at least as strong (as deemed by the identity provider) as one of the authentication contexts
848 specified.

849 If `<AuthnContextComparison>` is specified and set to *better*, then the resulting authentication statement in the
850 assertion (if any) MUST be stronger (as deemed by the identity provider) than any specified in the supplied
851 authentication contexts.

852 If `<AuthnContextComparison>` is specified and set to *maximum*, then the resulting authentication statement in the
853 assertion (if any) MUST be as strong as possible (as deemed by the identity provider) without exceeding the strength
854 of at least one of the authentication contexts specified.

855 If the identity provider wishes to rely on a second identity provider as the source of the Principal's authentication, then
856 the provider MUST follow the rules specified in [Section 3.2.2.7](#).

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

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

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

869 **3.2.2.7. Dynamic Proxying of Identity Providers**

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

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

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

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

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

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

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

- 885 When creating the new authentication request:
- 886 The identity provider **MUST** include equivalent or stricter forms of all the information included in the original
887 authentication request (such as authentication context policy).
- 888 If the authenticating provider is not a Liberty provider that implements this specifications, then the proxying provider
889 **MUST** have some other way to ensure that the elements governing Principal interaction (`<IsPassive>`, for example)
890 will be honored by the authenticating provider.
- 891 The new request **MUST** contain a `<ProxyCount>` element with a value of at least one less than the original
892 value. If the original request does not contain a `<ProxyCount>` element, then the new request **SHOULD** contain
893 a `<ProxyCount>` element.
- 894 If an `<IDPList>` was specified in the original request, the new request **MUST** also contain an `<IDPList>`.
- 895 The identity provider **MAY** add additional identity providers to the end of `<IDPList>`, but **MUST NOT** remove
896 providers from the list.
- 897 The authentication request and response are processed in normal fashion, in accordance with the rules given in
898 [Section 3.2](#). Once the Principal has authenticated to the proxying identity provider, the following steps are followed:
- 899 The proxying identity provider prepares a new authentication assertion on its own behalf by copying in the relevant in-
900 formation from the original assertion. The original assertion will be restricted by `AudienceRestrictionCondition`
901 to (at least) the identity provider, while the new assertion's condition will reference (at least) the original requesting
902 provider.
- 903 If the `<NameIdentifier>` has one-time semantics (determined by examining the `Format`), then the identity provider
904 **MUST** generate a new one-time identifier and include it in the new assertion.
- 905 If the `<NameIdentifier>` is not one-time, then the identity provider **MUST** include the Principal's fed-
906 erated `<IDPProvidedNameIdentifier>` for the requesting provider or affiliation group, as well as the
907 `<NameIdentifier>` provided by the requesting provider or affiliation group, if any.
- 908 If the identity provider does rely on a second provider to authenticate the principal, then its response to the original
909 requester **MUST** include an `<AuthnContext>` element containing an `<ac:AuthenticatingAuthority>` element
910 referencing the identity provider to which the responding provider referred the Principal.
- 911 If the original assertion contains `<AuthnContext>` information that includes one or more `<ac:AuthenticatingAuthority>`
912 elements, those elements **SHOULD** be included in the new assertion, with the new element placed after them.
- 913 Any other `<AuthnContext>` information **MAY** be copied, translated, or omitted in accordance with the policies of
914 the identity provider, provided that the original requirements dictated by the requesting provider are met.
- 915 If the authenticating identity provider is not a Liberty provider that implements the ID-FF specifications, then the
916 proxying identity provider **MUST** generate a `ProviderID` value for the authenticating provider. This value **SHOULD**
917 be consistent over time across different requests. The value **MUST** not conflict with values used or generated by other
918 Liberty providers.
- 919 If, in the future the identity provider is asked to authenticate the same Principal for a second provider, and this
920 provider's request is equally or less strict than the original provider's request, the identity provider **MAY** skip the
921 creation of a new request to the authenticating identity provider. The concrete definition of "less strict" and "equivalent"
922 is up to the identity provider, following the guidelines in section 3.2.3.
- 923 **3.2.2.8. Active Intermediaries**

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

927 NOTE: an active intermediary has the capability to return status codes to the service provider it interacts with. For
928 example, the intermediary may be unable to contact an identity provider identified by the service provider, and the
929 intermediary may return a status code to the service provider indicating that an error occurred. Status codes MUST be
930 conveyed within `<AuthnResponse>` messages using the `<samlp:Status>` element, according to the rules specified
931 in [\[SAMLCORE11\]](#), utilizing second-, and lower-level `<samlp:StatusCode>` elements. Specific values are defined
932 below. Service providers should also note that intermediaries are not providers, and hence may not have clocks
933 as accurately synchronized. This may invalidate the `IssueInstant` attribute included in the `<AuthnResponse>`
934 received by the service provider.

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

936 Specify whether the `<AuthnRequest>` element sent from the service provider to the identity provider via the
937 intermediary is wrapped in an `<AuthnRequestEnvelope>`. See [Section 3.2.3](#).

938 Specify whether the `<AuthnResponse>` element sent from the identity provider to the service provider via the
939 intermediary is wrapped in an `<AuthnResponseEnvelope>`. See [Section 3.2.4](#).

940 **3.2.2.8.1. Processing Rules for Active Intermediaries**

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

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

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

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

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

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

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

961 **3.2.2.9. Status Code Values for Error Conditions**

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

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

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

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

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

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

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

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

979 **3.2.3. Request Envelope**

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

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

986 **3.2.3.1. Element <AuthnRequestEnvelope>**

987 The authentication request envelope contains the following elements:

988 *Extension* [Optional]

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

992 *AuthnRequest* [Required]

993 The authentication request contained within the envelope.

994 *ProviderID* [Required]

995 The requester's ProviderID.

996 *ProviderName* [Optional]

997 The human-readable name of the requester.

998 *AssertionConsumerServiceURL* [Required]

999 A URL specifying where <AuthnResponse> elements, locally generated by an intermediary, should be sent.
1000 See the processing rules for active intermediaries specified in section 3.2.3.1.1.

1001 *IDPList* [Optional]

1002 A list of identity providers, from which, one may be chosen to service the authentication request.

1003 `IsPassive` [Optional]
1004 If "true," specifies that any intermediary between the service provider and identity provider MUST NOT
1005 interact with the Principal. If not specified, "true" is presumed.

1006 The schema fragment is as follows:

```
1007 <xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType" />
1008 <xs:complexType name="AuthnRequestEnvelopeType">
1009   <xs:complexContent>
1010     <xs:extension base="RequestEnvelopeType">
1011       <xs:sequence>
1012         <xs:element ref="AuthnRequest" />
1013         <xs:element ref="ProviderID" />
1014         <xs:element name="ProviderName" type="xs:string" minOccurs="0" />
1015         <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI" />
1016         <xs:element ref="IDPList" minOccurs="0" />
1017         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0" />
1018       </xs:sequence>
1019     </xs:extension>
1020   </xs:complexContent>
1021 </xs:complexType>
1022 <xs:complexType name="RequestEnvelopeType">
1023   <xs:sequence>
1024     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1025   </xs:sequence>
1026 </xs:complexType>
1027
1028
```

1029 3.2.3.2. Element `<IDPList>`

1030 In the request envelope, some profiles may wish to allow the service provider to transport a list of identity providers
1031 to the user agent. This specification provides a schema that profiles SHOULD use for this purpose. The elements are
1032 as follows:

1033 `IDPList` [Optional]

1034 The container element for an IDP List.

1035 `IDPEntries` [Required]

1036 Contains a list of identity provider entries.

1037 `IDPEntry` [Required]

1038 Describes an identity provider that the service provider supports.

1039 `ProviderID` [Required]

1040 The identity provider's unique identifier.

1041 `ProviderName` [Optional]

1042 The identity provider's human-readable name.

1043 `Loc` [Optional]

1044 The identity provider's URI, to which authentication requests may be sent. If present, this MUST be set
1045 to the value of the identity provider's `<SingleSignOnService>` element, obtained from their metadata
1046 ([LibertyMetadata](#)).

1047 `GetComplete` [Optional]

1048 If the identity provider list is not complete, this element may be included with a URI that points to where the
1049 complete list can be retrieved.

1050 The schema fragment is as follows:

```

1051 <xs:element name="IDPList" type="IDPListType"/>
1052 <xs:complexType name="IDPListType">
1053   <xs:sequence>
1054     <xs:element ref="IDPEntries"/>
1055     <xs:element ref="GetComplete" minOccurs="0"/>
1056   </xs:sequence>
1057 </xs:complexType>
1058 <xs:element name="IDPEntry">
1059   <xs:complexType>
1060     <xs:sequence>
1061       <xs:element ref="ProviderID"/>
1062       <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
1063       <xs:element name="Loc" type="xs:anyURI"/>
1064     </xs:sequence>
1065   </xs:complexType>
1066 </xs:element>
1067 <xs:element name="IDPEntries">
1068   <xs:complexType>
1069     <xs:sequence>
1070       <xs:element ref="IDPEntry" maxOccurs="unbounded"/>
1071     </xs:sequence>
1072   </xs:complexType>
1073 </xs:element>
1074 <xs:element name="GetComplete" type="xs:anyURI"/>
1075
1076

```

1077 3.2.3.3. Example

```

1078
1079     <AuthnRequestEnvelope>
1080     <AuthnRequest> ... </AuthnRequest>
1081     <ProviderID>http://ServiceProvider.com</ProviderID>
1082     <ProviderName>Service Provider X</ProviderName>
1083     <AssertionConsumerServiceURL>http://ServiceProvider.com/lecp_assertion_consume
1084 r</AssertionConsumerServiceURL>
1085     <IDPList>
1086     <IDPEntries>
1087     <IDPEntry>
1088     <ProviderID>http://IdentityProvider.com</ProviderID>
1089     <ProviderName>Identity Provider X</ProviderName>
1090     <Loc>http://www.IdentityProvider.com/liberty/ssl</Loc>
1091     </IDPEntry>
1092     </IDPEntries>
1093     <GetComplete>https://ServiceProvider.com/idplist?id=604be136-fe91-441e-afb8-f887
1094 48ae3b8b </GetComplete>
1095     </IDPList>
1096     <IsPassive>0</IsPassive>
1097     </AuthnRequestEnvelope>

```

1098 3.2.4. Response Envelope

1099 As with the <AuthnRequest> element, some profiles MAY wrap the <AuthnResponse> element in an envelope.
1100 This envelope allows for extra processing by an intermediary (such as a user agent or proxy) between the identity
1101 provider and the service provider. Applicable processing rules are given in section 3.2.3.3.1. Note that the envelope
1102 is for consumption by the intermediary and is removed prior to the forwarding of the enveloped <AuthnResponse>
1103 element to the service provider.

1104 3.2.4.1. Element <AuthnResponseEnvelope>

1105 The authentication response envelope contains the following elements:

1106 `Extension` [Optional]
1107 Optional container for protocol extensions established by agreement between service providers and interme-
1108 diaries. Implementers should note that this element may not contain content from the core Liberty namespace
1109 (which is prevented at the schema level by requiring `namespace="##other"`).

1110 `AuthnResponse` [Required]
1111 The enveloped authentication response.

1112 `AssertionConsumerServiceURL` [Required]
1113 The service provider's URL where the authentication response should be sent. This element's value SHOULD
1114 be obtained from the element of the same name in the service provider's metadata (see [\[LibertyMetadata\]](#)).

1115 The schema fragment is as follows:

```
1116 <xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType" />
1117 <xs:complexType name="AuthnResponseEnvelopeType">
1118   <xs:complexContent>
1119     <xs:extension base="ResponseEnvelopeType">
1120       <xs:sequence>
1121         <xs:element ref="AuthnResponse" />
1122         <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI" />
1123       </xs:sequence>
1124     </xs:extension>
1125   </xs:complexContent>
1126 </xs:complexType>
1127 <xs:complexType name="ResponseEnvelopeType">
1128   <xs:sequence>
1129     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1130   </xs:sequence>
1131 </xs:complexType>
1132
1133
```

1134 3.2.4.2. Example

```
1135 <AuthnResponseEnvelope>
1136   <AuthnResponse> ... </AuthnResponse>
1137   <AssertionConsumerServiceURL>
1138     http://ServiceProvider.com/lecp_assertion_consumer
1139   </AssertionConsumerServiceURL>
1140 </AuthnResponseEnvelope>
```

1142 **3.3. Name Registration Protocol**

1143 During federation, the identity provider generates an opaque value that serves as the initial name identifier that both
1144 the service provider and the identity provider use in referring to the Principal when communicating with each other.
1145 This name identifier is termed the `<IDPProvidedNameIdentifier>`.

1146 Subsequent to federation, the service provider MAY register a different opaque value with the identity provider. This
1147 opaque value is termed the `<SPProvidedNameIdentifier>`. Until the service provider registers a different name,
1148 the identity provider will use `<IDPProvidedNameIdentifier>` to refer to the Principal when communicating with
1149 the service provider.

1150 After a service provider's name registration, the identity provider MUST use the `<SPProvidedNameIdentifier>`
1151 for `<saml:NameIdentifier>` elements when communicating to the service provider about the Principal.
1152 The service provider MUST use the current (most recently supplied) `<IDPProvidedNameIdentifier>` for
1153 `<saml:NameIdentifier>` elements when communicating to the identity provider about the Principal.

1154 Either the service provider or the identity provider MAY register a new name identifier for a Principal with each
1155 other at any time following federation. The name identifiers specified by providers SHOULD adhere to the following
1156 guidelines:

- 1157 • The name identifier SHOULD be unique across the identity providers with which the Principal's identity is
1158 federated.
- 1159 • The name identifier SHOULD be unique within the group of name identifiers that have been registered with the
1160 identity provider by this service provider.
- 1161 • Only federated identifiers (and by extension pre-1.2 identifiers which are always federated) can be replaced and
1162 set with this protocol. Encrypted or one-time identifiers MUST NOT be used.
- 1163 • The `Format` attribute in the newly registered identifier element MUST be `urn:liberty:iff:nameid:federated`.
1164 The `Format` attribute in the `<OldProvidedNameIdentifier>` element MUST either be
1165 `urn:liberty:iff:nameid:federated`, or MAY be omitted or set to a non-standard value if a pre-1.2
1166 federation is being updated. This also applies to the `<SPProvidedNameIdentifier>` sent by an IdP or the
1167 `<IDPProvidedNameIdentifier>` sent by an SP if referencing a pre-1.2 federation.
- 1168 • Similarly, the `NameQualifier` attribute in the newly registered identifier MUST be the service provider's unique
1169 identifier, an affiliation group's unique identifier, or omitted (implying the service provider's identifier). The other
1170 elements either follow the same rules, or if a pre-1.2 federation, MUST be treated as literal data.

1171 **3.3.1. Request**

1172 To register a `<SPProvidedNameIdentifier>` with an identity provider, the service provider sends a
1173 `<RegisterNameIdentifierRequest>` message.

1174 The same `<RegisterNameIdentifierRequest>` message may be sent by an identity provider, seeking to change
1175 the `<IDPProvidedNameIdentifier>` stored by the service provider.

1176 The `<RegisterNameIdentifierRequest>` message SHOULD be signed.

1177 **3.3.1.1. Element `<RegisterNameIdentifierRequest>`**

1178 The elements of the message are as follows:

- 1179 Extension [Optional]
1180 Optional container for protocol extensions established by agreement between providers.
- 1181 ProviderID [Required]
1182 The provider's identifier.
- 1183 IDPProvidedNameIdentifier [Required]
1184 For an IdP-initiated request, the new name identifier the service provider should use when communicating
1185 with the identity provider. For an SP-initiated request, the current name identifier established by the IdP for
1186 the SP to use when communicating with it.
- 1187 SPPProvidedNameIdentifier [Optional]
1188 For an SP-initiated request, this is required, and is the new name identifier the identity provider should use
1189 when communicating to the service provider. For an IdP- initiated request, this is the current name identifier
1190 established by the SP for the IdP to use when communicating with it, or if none exists, MUST be omitted.
- 1191 OldProvidedNameIdentifier [Required]
1192 In the case of either provider choosing to request a change of provided name identifiers, this element holds the
1193 previous version set by that provider. For a service provider making their first name change following feder-
1194 ation, the <OldProvidedNameIdentifier> will contain the current <IDPProvidedNameIdentifier>.
- 1195 RelayState [Optional]
1196 This contains state information that will be relayed back in the response. This data SHOULD be integrity-
1197 protected by the request author and MAY have other protections placed on it by the request author. An
1198 example of such protection is confidentiality.

1199 The schema fragment is as follows:

```
1200 <xs:element name="RegisterNameIdentifierRequest" type="RegisterNameIdentifierRequestType"/>
1201 <xs:complexType name="RegisterNameIdentifierRequestType">
1202   <xs:complexContent>
1203     <xs:extension base="samlp:RequestAbstractType">
1204       <xs:sequence>
1205         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1206         <xs:element ref="ProviderID"/>
1207         <xs:element ref="IDPProvidedNameIdentifier"/>
1208         <xs:element ref="SPPProvidedNameIdentifier" minOccurs="0"/>
1209         <xs:element ref="OldProvidedNameIdentifier"/>
1210         <xs:element ref="RelayState" minOccurs="0"/>
1211       </xs:sequence>
1212     </xs:extension>
1213   </xs:complexContent>
1214 </xs:complexType>
1215 <xs:element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1216 <xs:element name="SPPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1217 <xs:element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1218
1219
```

1220 3.3.1.2. Example

```
1221
1222 <RegisterNameIdentifierRequest RequestID="eb20e77f-d982-44f9-936e-dd135bf437d4"
1223   MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">
1224   <ds:Signature>...</ds:Signature>
1225   <ProviderID>http://ServiceProvider.com</ProviderID>
1226   <IDPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
1227     Format="urn:liberty:iff:nameid:federated">342ad3d8-93e
1228 e-4c68-be35-cc9e7db39e2b</IDPProvidedNameIdentifier>
1229   <SPPProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
1230     Format="urn:liberty:iff:nameid:federated">e958019a</S
1231 PProvidedNameIdentifier>
```



```
1232         <OldProvidedNameIdentifier NameQualifier="http://ServiceProvider.com"
1233             Format="urn:liberty:iff:nameid:federated">e8950
1234 14a</OldProvidedNameIdentifier>
1235         <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1236     </RegisterNameIdentifierRequest>
```

1237 3.3.2. Response

1238 The recipient **MUST** respond with a `<RegisterNameIdentifierResponse>` message, which is of
1239 type **StatusResponseType**. **StatusResponseType** is an extension of **samlp:ResponseType** and a
1240 `<samlp:Status>` element and a `<RelayState>` may exist in the body.

1241 This message **SHOULD** be signed.

1242 3.3.2.1. Element `<RegisterNameIdentifierResponse>`

1243 The elements of the message are as follows:

1244 Extension [Optional]

1245 Optional container for protocol extensions established by agreement between providers.

1246 ProviderID [Required]

1247 The provider's unique identifier.

1248 Status [Required]

1249 The status of the request processing.

1250 RelayState [Optional]

1251 This element contains state information that will be relayed back in the response, if it has been supplied in
1252 the request.

1253 The schema fragment is as follows:

```
1254 <xs:element name="RegisterNameIdentifierResponse" type="StatusResponseType" />
1255 <xs:complexType name="StatusResponseType">
1256     <xs:complexContent>
1257         <xs:extension base="samlp:ResponseAbstractType">
1258             <xs:sequence>
1259                 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1260                 <xs:element ref="ProviderID" />
1261                 <xs:element ref="samlp:Status" />
1262                 <xs:element ref="RelayState" minOccurs="0" />
1263             </xs:sequence>
1264         </xs:extension>
1265     </xs:complexContent>
1266 </xs:complexType>
1267
1268
```

1269 3.3.2.2. Example

```
1270
1271     <RegisterNameIdentifierResponse ResponseID="ff74ec0f-1165-4fa3-b0
1272 88-3dd2c2388b91"
1273         InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
1274         MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
1275         Recipient="http://ServiceProvider.com">
1276         <ds:Signature>...</ds:Signature>
1277         <ProviderID>http://ServiceProvider.com</ProviderID>
1278         <samlp:Status>
1279             <samlp:StatusCode Value="samlp:Success" />
```

```
1280         </samlp:Status>  
1281         <RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>  
1282     </RegisterNameIdentifierResponse>
```

1283 3.3.3. Processing Rules

1284 The recipient **MUST** validate any signature present on the message. To be considered valid, the signature provided
1285 **MUST** be the signature of the <ProviderID> contained in the message.

1286 If the request includes an <IDPProvidedNameIdentifier> for which no federation exists between the service
1287 provider and the identity provider, the identity provider **MUST** respond with a <samlp:Status> element containing
1288 a second-level <samlp:StatusCode> of lib:FederationDoesNotExist. Otherwise, the identity provider **MUST** use
1289 <SPProvidedNameIdentifier> when subsequently communicating to the service provider regarding this Principal.

1290 Either provider **MAY** choose to change their provided name identifier. In this case, the <OldProvidedNameIdentifier>
1291 should contain the previous version of their name identifier. When a service provider chooses to
1292 change their provided name identifier, the <OldProvidedNameIdentifier> should contain the current
1293 <SPProvidedNameIdentifier>. Note that when they first change their name, this will be equal to the
1294 <IDPProvidedNameIdentifier>. Similarly, when an identity provider wishes to change their provided name
1295 identifier, they will move the previous version to the <OldProvidedNameIdentifier> when sending this message.

1296 In all of the name identifier elements in the request and response messages of this protocol, if the Principal's identity
1297 federation is between the identity provider and an affiliation group in which the service provider is a member, then the
1298 NameQualifier attribute **MUST** contain the unique identifier of the affiliation group. Otherwise, it **MUST** contain
1299 the unique identifier of the service provider or **MAY** be omitted (implying the service provider's identifier). The
1300 exception as noted previously is if the old identifier(s) represent a pre-1.2 federation being updated with the request,
1301 in which case the attribute is simply treated as literal data or omitted.

1302 Changes to these identifiers may take a potentially significant amount of time to propagate through the systems at both
1303 the sender and the receiver. Implementations may wish to allow each party to accept either identifier for some period
1304 of time following the successful completion of a name identifier change. Not doing so could result in the inability of
1305 the Principal to access resources.

1306 If <RelayState> contains a value, the recipient **MUST** include this value in unmodified form in the <RelayState>
1307 element of the response.

1308 3.4. Federation Termination Notification Protocol

1309 When the Principal terminates an identity federation between a service provider and an identity provider from the
1310 service provider, the service provider **MUST** send a <FederationTerminationNotification> message to the
1311 identity provider. The service provider is stating that it will no longer accept authentication assertions from the identity
1312 provider for the specified Principal.

1313 Likewise, when the Principal terminates an identity federation from the identity provider, the identity provider **MUST**
1314 send a <FederationTerminationNotification> message to the service provider. In this case, the identity
1315 provider is stating that it will no longer provide authentication assertions to the service provider for the specified
1316 Principal.

1317 This notification message is a one-way asynchronous message. Reasonable, best-effort delivery **MUST** be employed
1318 by all providers sending this message.

1319 3.4.1. Message

1320 The provider sends a <FederationTerminationNotification> to the provider with which it is terminating a
1321 federation.

1322 The <FederationTerminationNotification> message SHOULD be signed.

1323 3.4.1.1. Element <FederationTerminationNotification>

1324 The elements are as follows:

1325 Extension [Optional]

1326 Optional container for protocol extensions established by agreement between providers.

1327 ProviderID [Required]

1328 The identifier of the provider that is sending this message.

1329 NameIdentifier [Required]

1330 The name identifier of the Principal terminating federation. This name identifier MUST be equal to
1331 the <saml:NameIdentifier> element (and its included attributes) agreed upon earlier between the two
1332 communicating providers.

1333 consent [Optional]

1334 Indicates whether or not consent has been obtained from a user in sending this message.

1335 The schema fragment is as follows:

```
1336 <xs:element name="FederationTerminationNotification" type="FederationTerminationNotif
1337 icationType"/>
1338 <xs:complexType name="FederationTerminationNotificationType">
1339   <xs:complexContent>
1340     <xs:extension base="samlp:RequestAbstractType">
1341       <xs:sequence>
1342         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1343         <xs:element ref="ProviderID"/>
1344         <xs:element ref="saml:NameIdentifier"/>
1345       </xs:sequence>
1346       <xs:attribute ref="consent" use="optional"/>
1347     </xs:extension>
1348   </xs:complexContent>
1349 </xs:complexType>
1350
1351
```

1352 3.4.1.2. Example

```
1353
1354 <FederationTerminationNotification RequestID="e9c2-eb65-4bce-ab8f-4becdf229815"
1355   MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
1356   IssueInstant="2001-12-17T09:30:47Z">
1357   <ds:Signature>...</ds:Signature>
1358   <ProviderID>http://IdentityProvider.com</ProviderID>
1359   <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
1360     Format="urn:liberty:iff:nameid:federated">e958019a</saml:NameIdentifier>
1361   <RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1362 </FederationTerminationNotification>
```

1363 3.4.2. Processing Rules

1364 The receiving provider MUST validate any signature present on the message. The signature on the message MUST be
1365 the signature of the <ProviderID> contained in the message. If the signature is not valid, the provider MUST ignore
1366 the message.

1367 If a provider receives a federation termination notification message that refers to a federation that does not exist from
1368 the perspective of the provider, the provider **MUST** ignore the message. Otherwise, the provider **MAY** perform any
1369 maintenance with the knowledge that the federation has been terminated.

1370 A provider **MAY** choose to invalidate the session of a user for whom federation has been terminated.

1371 If the Principal's identity federation was between the identity provider and an affiliation group in which the service
1372 provider is a member, then the `NameQualifier` attribute **MUST** contain the unique identifier of the affiliation group.
1373 Otherwise, it **MUST** contain the unique identifier of the service provider or **MAY** be omitted (implying the service
1374 provider's identifier). The exception is a pre-1.2 federated identifier (recognizable by a missing or non-standard
1375 `Format` attribute), in which case the `NameQualifier` attribute is simply treated as literal data or omitted.

1376 **3.5. Single Logout Protocol**

1377 The Single Logout Protocol provides a message exchange protocol by which all sessions authenticated by a particular
1378 identity provider are near-simultaneously terminated. The Single Logout Protocol is used either when a Principal logs
1379 out at a service provider or when the Principal logs out at an identity provider.

1380 The Principal may have established authenticated sessions both with the identity provider, and individual service
1381 providers, based on authentication assertions supplied by the identity provider.

1382 When the Principal invokes the single logout process at a service provider, the service provider **MUST** send a
1383 `<LogoutRequest>` message to the identity provider that provided the authentication service related to that session at
1384 the service provider.

1385 When either the Principal invokes a logout at the identity provider or a service provider sends a logout request to the
1386 identity provider specifying that Principal, the identity provider **MUST** send a `<LogoutRequest>` message to each
1387 service provider to which it provided authentication assertions under its current session with the Principal, with the
1388 exception of the service provider that sent the `<LogoutRequest>` message to the identity provider.

1389 The following rules apply to identity providers involved in authentication proxying:

1390 • If the identity provider is proxying authentication from a second identity provider, then it **MUST** send a
1391 `<LogoutRequest>` to the proxied identity provider, unless the proxying provider has already received a
1392 `<LogoutRequest>` from the proxied provider.

1393 • If the identity provider has provided authentication assertions on behalf of a Principal to a proxying identity
1394 provider, then it **MUST** send a `<LogoutRequest>` to that provider, unless the proxying provider has already
1395 received a `<LogoutRequest>` from the proxied provider.

1396 **3.5.1. Request**

1397 The <LogoutRequest> message indicates to the message receiver that a Principal's session was terminated. The
1398 message includes an optional *SessionIndex* element that **MUST** be specified if and only if a *SessionIndex* element
1399 was present in the authentication statement in the assertion that the service provider used in establishing the session
1400 with the Principal. This message **SHOULD** be signed.

1401 **3.5.1.1. Element <LogoutRequest>**

1402 *Extension* [Optional]

1403 Optional container for protocol extensions established by agreement between providers. Implementers should
1404 note that this element may not contain content from the core Liberty namespace (which is prevented at the
1405 schema level by requiring `namespace="##other"`).

1406 *NameIdentifier* [Required]

1407 The name identifier of the Principal that logged out. This name identifier **MUST** be equal to the
1408 <saml:*NameIdentifier*> element (including the equality of contained attributes) agreed upon between
1409 the two communicating providers.

1410 *ProviderID* [Required]

1411 The identifier of the provider that is making the request.

1412 *SessionIndex* [Optional]

1413 The session index specified in the authentication statement of the assertion used to establish the ses-
1414 sion being terminated. If a <*SessionIndex*> element was present in the authentication statement, an
1415 identical <*SessionIndex*> **MUST** be present in the <LogoutRequest>. If no <*SessionIndex*> el-
1416 ement was present in the authentication statement, the <*SessionIndex*> **MUST** be omitted from the
1417 <LogoutRequest>.

1418 *RelayState* [Optional]

1419 This may contain state information that will be relayed back in the response. This data **SHOULD** be integrity-
1420 protected by the request author and **MAY** have other protections placed on it by the request author. An
1421 example of such protection is confidentiality.

1422 *consent* [Optional]

1423 Indicates whether or not consent has been obtained from a user in sending this message.

1424 *NotOnOrAfter* [Optional]

1425 This attribute is used to specify the time instant at which the recipient may expire a logout request from
1426 the sender. The sender **MAY** set this value equal to the *NotOnOrAfter* attribute of a previously-supplied
1427 assertion, if one was specified.

1428 The schema fragment is as follows:

```
1429 <xs:element name="LogoutRequest" type="LogoutRequestType" />
1430 <xs:complexType name="LogoutRequestType">
1431   <xs:complexContent>
1432     <xs:extension base="samlp:RequestAbstractType">
1433       <xs:sequence>
1434         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1435         <xs:element ref="ProviderID" />
1436         <xs:element ref="saml:NameIdentifier" />
1437         <xs:element name="SessionIndex" type="xs:string" minOccurs="0"
1438 maxOccurs="unbounded" />
1439         <xs:element ref="RelayState" minOccurs="0" />
1440       </xs:sequence>
1441       <xs:attribute ref="consent" use="optional" />
1442       <xs:attribute name="NotOnOrAfter" type="xs:dateTime" use="optional" />
```

```
1443         </xs:extension>
1444     </xs:complexContent>
1445 </xs:complexType>
1446
1447
```

1448 3.5.1.2. Example

```
1449
1450     <LogoutRequest RequestID="d4769303-7c33-4d65-931f-ddeb19fa6a73"
1451       MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
1452       IssueInstant="2001-12-17T09:30:47Z">
1453       <ds:Signature>...</ds:Signature>
1454       <ProviderID>http://ServiceProvider.com</ProviderID>
1455       <saml:NameIdentifier NameQualifier="http://ServiceProvider.com"
1456         Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be3
1457 5-cc9e7db39e2b</saml:NameIdentifier>
1458       <SessionIndex>3</SessionIndex>
1459       <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1460     </LogoutRequest>
```

1461 3.5.2. Response

1462 The recipient **MUST** return a <LogoutResponse> message, which is of type StatusResponseType.

1463 This message **SHOULD** be signed.

1464 3.5.2.1. Element <LogoutResponse>

1465 The elements of the message are as follows:

1466 **Extension** [Optional]

1467 Optional container for protocol extensions established by agreement between providers.

1468 **ProviderID** [Required]

1469 The identifier of the provider responding.

1470 **Status** [Required]

1471 A status code that indicates the result of the request.

1472 **RelayState** [Optional]

1473 This contains state information that may have appeared in the request, and is being relayed back to the sender.

1474 The schema fragment is as follows:

```
1475     <xs:element name="LogoutResponse" type="StatusResponseType" />
1476
1477
```

1478 3.5.2.2. Example

```
1479
1480     <LogoutResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
1481       InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
1482       MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
1483       Recipient="http://ServiceProvider.com">
1484       <ds:Signature>...</ds:Signature>
1485       <ProviderID>http://IdentityProvider.com</ProviderID>
1486       <samlp:Status>
1487         <samlp:StatusCode Value="samlp:Success" />
1488     </samlp:Status>
```

```
1489         <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZt uMFQxDS8b</RelayState>  
1490     </LogoutResponse>
```

1491 3.5.2.3. Processing Rules

1492 If `<RelayState>` contains a value, the recipient **MUST** include this value in unmodified form in the `<RelayState>`
1493 element of the response.

1494 If the Principal's identity federation was between the identity provider and an affiliation group in which the service
1495 provider is a member, then the `NameQualifier` attribute **MUST** contain the unique identifier of the affiliation group.
1496 Otherwise, it **MUST** contain the unique identifier of the service provider or **MAY** be omitted (implying the service
1497 provider's identifier). The exception is a pre-1.2 federated identifier (recognizable by a missing or non-standard
1498 `Format` attribute), in which case the `NameQualifier` attribute is simply treated as literal data or omitted.

1499 Other unique processing rules apply based on whether the message receiver is an identity provider or a service provider.

1500 3.5.2.3.1. Identity Provider Processing Rules

1501 When an identity provider receives the `<LogoutRequest>` message, the identity provider **MUST** validate that any
1502 signature present on the message is the signature of a service provider to which the identity provider provided an
1503 authentication assertion for the current session. If that holds, the identity provider **SHOULD** do the following:

- 1504 • Send a `<LogoutRequest>` message to each service provider for which the identity provider provided authentica-
1505 tion assertions in the current session, other than the originator of the `<LogoutRequest>`.
- 1506 • Send a `<LogoutRequest>` message to the identity provider on behalf of whom the identity provider proxied the
1507 user's authentication, unless the second identity provider is the originator of the `<LogoutRequest>`.
- 1508 • Terminate the Principal's current session as specified by the `<saml:NameIdentifier>` element, and any
1509 `<SessionIndex>` present in the logout request message.

1510 When constructing a logout request message, the identity provider **MUST** set the value of the `NotOnOrAfter` attribute
1511 of the message to a time value, indicating an expiration time for the message. See [\[LibertyImplGuide\]](#) for more details.

1512 If an error occurs during this further processing of the logout (for example, relying service providers may not all
1513 implement the Single Logout profile used by the requesting service provider), then the identity provider **MUST** respond
1514 to the original requester with a `<LogoutResponse>` message, indicating the status of the logout request. The value
1515 "lib:UnsupportedProfile" is provided for a second-level `<samlp:StatusCode>`, indicating that a service provider
1516 should retry the `<LogoutRequest>` using a different profile.

1517 3.5.2.3.2. Service Provider Processing Rules

1518 When the service provider receives the `<LogoutRequest>` message, the service provider **MUST** validate the identity
1519 provider's signature contained in the `<ds:Signature>` element. If the signature is that of the identity provider that
1520 provided the authentication for the Principal's current session, the service provider **MUST** invalidate the Principal's
1521 session(s) referred to by the `<saml:NameIdentifier>` element, and any `SessionIndex` elements supplied in the
1522 message.

1523 The service provider **MUST** apply the logout request message to any assertion that meets the following conditions,
1524 *even if the assertion arrives after the logout request:*

- 1525 1. The `SessionIndex` of the assertion matches one specified in the logout request.
- 1526 2. The assertion would otherwise be valid

1527 3. The logout request has not yet expired (determined by examining the `NotOnOrAfter` attribute on the message).

1528 When constructing a logout request message, the service provider SHOULD omit the *NotOnOrAfter* attribute.

1529 3.6. Name Identifier Mapping Protocol

1530 When a service provider requires a name identifier for a Principal with which it has an identity federation relationship,
1531 but which references an identity federation between the identity provider and another service provider, it can use this
1532 protocol to obtain such an identifier. This allows the requesting provider to communicate with the other service
1533 provider about the Principal without an identity federation for the Principal between them. The resulting value
1534 SHOULD be encrypted so as to obscure the actual value from anyone but the second service provider. To the requester,
1535 it will be an opaque (and one-time) value.

1536 Upon receipt of a `<NameIdentifierMappingRequest>` message, an identity provider that supports this protocol
1537 MUST respond with a `<NameIdentifierMappingResponse>` message.

1538 3.6.1. Request

1539 The requesting service provider sends a `<NameIdentifierMappingRequest>` message to the identity provider
1540 which can provide the desired federated name identifier.

1541 The `<NameIdentifierMappingRequest>` message MUST be signed.

1542 3.6.1.1. Element `<NameIdentifierMappingRequest>`

1543 The elements and attributes are as follows:

1544 `Extension` [Optional]

1545 Optional container for protocol extensions established by agreement between providers.

1546 `ProviderID` [Required]

1547 The unique identifier of the provider that is sending this message.

1548 `saml:NameIdentifier` [Required]

1549 The name identifier of the Principal for whom the requester is obtaining a mapped identifier. See the
1550 processing rules below for additional information on the content of this element.

1551 `TargetNamespace` [Required]

1552 The unique identifier of the service provider or affiliation group for whom the requester needs the name
1553 identifier of the Principal to subsequently communicate with.

1554 `consent` [Optional]

1555 Indicates whether or not consent has been obtained from a user in sending this message.

1556 The schema fragment is as follows:

```
1557 <xs:element name="NameIdentifierMappingRequest" type="NameIdentifierMappingRequestType" />
1558 <xs:complexType name="NameIdentifierMappingRequestType">
1559   <xs:complexContent>
1560     <xs:extension base="samlp:RequestAbstractType">
1561       <xs:sequence>
1562         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1563         <xs:element ref="ProviderID" />
1564         <xs:element ref="saml:NameIdentifier" />
1565         <xs:element name="TargetNamespace" type="md:entityIDType" />
1566       </xs:sequence>
1567       <xs:attribute ref="consent" use="optional" />
1568     </xs:extension>
```


1569 </xs:complexContent>
1570 </xs:complexType>
1571
1572

1573 3.6.1.2. Example

```
1574  
1575       <NameIdentifierMappingRequest RequestID="e9c2-eb65-4bce-ab8f-4becdf229815"  
1576       MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">  
1577       <ds:Signature>...</ds:Signature>  
1578       <ProviderID>http://RequestingServiceProvider.com</ProviderID>  
1579       <saml:NameIdentifier NameQualifier="http://RequestingServiceProvider.com"  
1580       Format="urn:liberty:iff:nameid:federated">e958019a</saml:NameIdentifier>  
1581       <TargetNamespace>http://TargetServiceProvider.com</TargetNamespace>  
1582     </NameIdentifierMappingRequest>
```

1583 3.6.2. Response

1584 The responding provider MUST return a <NameIdentifierMappingResponse> message if the signature is valid.

1585 This message SHOULD be signed.

1586 3.6.2.1. Element <NameIdentifierMappingResponse>

1587 The elements of the message are as follows:

1588 *Extension* [Optional]

1589 Optional container for protocol extensions established by agreement between providers.

1590 *ProviderID* [Required]

1591 The identifier of the provider responding.

1592 *Status* [Required]

1593 A status code that indicates the reception and processing status of the message.

1594 *saml:NameIdentifier* [Optional]

1595 If the request is successful, contains the resulting mapped name identifier for the desired identity federation,
1596 which SHOULD be in encrypted form. See the processing rules below for additional information on the
1597 content of this element.

1598 The schema fragment is as follows:

```
1599     <xs:element name="NameIdentifierMappingResponse" type="NameIdentifierMappingResponseType"/>  
1600     <xs:complexType name="NameIdentifierMappingResponseType">  
1601       <xs:complexContent>  
1602         <xs:extension base="samlp:ResponseAbstractType">  
1603           <xs:sequence>  
1604             <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>  
1605             <xs:element ref="ProviderID"/>  
1606             <xs:element ref="samlp:Status"/>  
1607             <xs:element ref="saml:NameIdentifier" minOccurs="0"/>  
1608           </xs:sequence>  
1609         </xs:extension>  
1610       </xs:complexContent>  
1611     </xs:complexType>  
1612  
1613
```

1614 3.6.2.2. Example

```
1615
1616         <NameIdentifierMappingResponse ResponseID="ff74ec0f-1165-4fa3-b088-3dd2c2388b91"
1617           InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"
1618           MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"
1619           Recipient="http://ServiceProvider.com">
1620         <ds:Signature>...</ds:Signature>
1621         <ProviderID>http://IdentityProvider.com</ProviderID>
1622         <samlp:Status>
1623           <samlp:StatusCode Value="samlp:Success"/>
1624         </samlp:Status>
1625         <saml:NameIdentifier NameQualifier="http://TargetServiceProvider.com"
1626           Format="urn:liberty:iff:nameid:federated">gfhfrfe89u43</saml:Name
1627 Identifier>
1628       </NameIdentifierMappingResponse>
```

1629 3.6.3. Processing Rules

1630 The receiving provider MUST validate any signature present on the message. The signature on the message MUST be
1631 the signature of the <ProviderID> contained in the message. If the signature is not valid, the provider MUST ignore
1632 the message.

1633 In the request message's <saml:NameIdentifier>, if the Principal's identity federation is between the identity
1634 provider and an affiliation group in which the requesting service provider is a member, then the NameQualifier
1635 attribute MUST contain the unique identifier of the affiliation group. Otherwise, it MUST contain the unique identifier
1636 of the service provider.

1637 3.6.3.1. Recipient Processing Rules

1638 Supporting identity providers MUST respond with a <NameIdentifierMappingResponse> message.

1639 If the identity provider is unable to identify the Principal referenced in the request, then it MUST return a response
1640 message with a second-level <samlp:StatusCode> Value of *lib:UnknownPrincipal*.

1641 If the identity provider is unable to map the name identifier of the Principal to a federation between it
1642 and the <TargetNamespace> in the request, then it MUST return a response message with a second-level
1643 <samlp:StatusCode> Value of *lib:FederationDoesNotExist*.

1644 If the mapping is successful, then the response message's <saml:NameIdentifier> MUST contain a
1645 NameQualifier that matches the request's <TargetNamespace> element. It SHOULD be a one-time en-
1646 crypted value but MAY be the actual federated identifier value; the Format attribute MUST indicate this distinction.

1647 4. Schema Definition

```
1648 <?xml version="1.0" encoding="UTF-8"?>
1649 <xs:schema targetNamespace="urn:liberty:iff:2003-08"
1650   xmlns="urn:liberty:iff:2003-08"
1651   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1652   xmlns:ac="urn:liberty:ac:2003-08"
1653   xmlns:md="urn:liberty:metadata:2003-08"
1654   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
1655   xmlns:sampl="urn:oasis:names:tc:SAML:1.0:protocol"
1656   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
1657   elementFormDefault="qualified" attributeFormDefault="unqualified">
1658
1659   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
1660     schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
1661   <xs:import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
1662     schemaLocation="oasis-sstc-saml-schema-protocol-1.1.xsd"/>
1663   <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
1664     schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>
1665   <xs:import namespace="urn:liberty:ac:2003-
1666 08" schemaLocation="liberty-authentication-context-v1.3.xsd" />
1667   <xs:import namespace="urn:liberty:metadata:2003-0
1668 8" schemaLocation="liberty-metadata-v1.1.xsd" />
1669
1670   <xs:include schemaLocation="liberty-idff-utility-v1.0.xsd" />
1671
1672   <xs:annotation>
1673     <xs:documentation> ### IMPORTANT NOTICE ###
1674
1675 The source code in this XSD file was excerpted verbatim from:
1676
1677 Liberty Protocols and Schema Specification
1678 Version 1.2-errata-v3.0
1679 19 May 2005
1680 Updated for WSF 1.1 release
1681
1682 Copyright (c) 2003-2005 Liberty Alliance participants, see
1683 http://www.projectliberty.org/specs/idff_copyrights.html
1684
1685 </xs:documentation>
1686 </xs:annotation>
1687 <xs:element name="ProviderID" type="md:entityIDType" />
1688 <xs:element name="AffiliationID" type="md:entityIDType" />
1689
1690 <xs:element name="AuthnRequest" type="AuthnRequestType" />
1691 <xs:complexType name="AuthnRequestType">
1692   <xs:complexContent>
1693     <xs:extension base="sampl:RequestAbstractType">
1694       <xs:sequence>
1695         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1696         <xs:element ref="ProviderID" />
1697         <xs:element ref="AffiliationID" minOccurs="0" />
1698         <xs:element ref="NameIDPolicy" minOccurs="0" />
1699         <xs:element name="ForceAuthn" type="xs:boolean" minOccurs="0" />
1700         <xs:element name="IsPassive" type="xs:boolean" minOccurs="0" />
1701         <xs:element ref="ProtocolProfile" minOccurs="0" />
1702         <xs:element name="AssertionConsumerServiceID" type="xs:string" minOccurs="0" />
1703         <xs:element ref="RequestAuthnContext" minOccurs="0" />
1704         <xs:element ref="RelayState" minOccurs="0" />
1705         <xs:element ref="Scoping" minOccurs="0" />
1706       </xs:sequence>
1707       <xs:attribute ref="consent" use="optional" />
1708     </xs:extension>
1709   </xs:complexContent>
1710 </xs:complexType>
1711 <xs:simpleType name="NameIDPolicyType">
1712   <xs:restriction base="xs:string">
```

```

1713         <xs:enumeration value="none" />
1714         <xs:enumeration value="onetime" />
1715         <xs:enumeration value="federated" />
1716         <xs:enumeration value="any" />
1717     </xs:restriction>
1718 </xs:simpleType>
1719 <xs:element name="NameIDPolicy" type="NameIDPolicyType" />
1720 <xs:simpleType name="AuthnContextComparisonType">
1721     <xs:restriction base="xs:string">
1722         <xs:enumeration value="exact" />
1723         <xs:enumeration value="minimum" />
1724         <xs:enumeration value="maximum" />
1725         <xs:enumeration value="better" />
1726     </xs:restriction>
1727 </xs:simpleType>
1728 <xs:complexType name="ScopingType">
1729     <xs:sequence>
1730         <xs:element name="ProxyCount" type="xs:nonNegativeInteger" minOccurs="0" />
1731         <xs:element ref="IDPList" minOccurs="0" />
1732     </xs:sequence>
1733 </xs:complexType>
1734 <xs:element name="Scoping" type="ScopingType" />
1735 <xs:element name="RelayState" type="xs:string" />
1736 <xs:element name="ProtocolProfile" type="xs:anyURI" />
1737 <xs:element name="RequestAuthnContext">
1738     <xs:complexType>
1739         <xs:sequence>
1740             <xs:choice>
1741                 <xs:element name="AuthnContextClassRef" type="xs:anyURI" maxOccurs="unbounded" />
1742                 <xs:element name="AuthnContextStatementRef" type="xs:anyURI" maxOccurs="unbounded" />
1743             </xs:choice>
1744             <xs:element name="AuthnContextComparison" type="AuthnContextComparisonType" ↵
1745             minOccurs="0" />
1746         </xs:sequence>
1747     </xs:complexType>
1748 </xs:element>
1749
1750 <xs:element name="AuthnResponse" type="AuthnResponseType" />
1751 <xs:complexType name="AuthnResponseType">
1752     <xs:complexContent>
1753         <xs:extension base="saml:ResponseType">
1754             <xs:sequence>
1755                 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1756                 <xs:element ref="ProviderID" />
1757                 <xs:element ref="RelayState" minOccurs="0" />
1758             </xs:sequence>
1759             <xs:attribute ref="consent" use="optional" />
1760         </xs:extension>
1761     </xs:complexContent>
1762 </xs:complexType>
1763 <xs:element name="Assertion" type="AssertionType" />
1764 <xs:complexType name="AssertionType">
1765     <xs:complexContent>
1766         <xs:extension base="saml:AssertionType">
1767             <xs:attribute name="InResponseTo" type="xs:NCName" use="optional" />
1768         </xs:extension>
1769     </xs:complexContent>
1770 </xs:complexType>
1771 <xs:complexType name="SubjectType">
1772     <xs:complexContent>
1773         <xs:extension base="saml:SubjectType">
1774             <xs:sequence>
1775                 <xs:element ref="IDPProvidedNameIdentifier" minOccurs="0" />
1776             </xs:sequence>
1777         </xs:extension>
1778     </xs:complexContent>
1779 </xs:complexType>

```

```

1780 <xs:element name="Subject" type="SubjectType" substitutionGroup="saml:Subject" />
1781 <xs:element name="EncryptableNameIdentifier" type="EncryptableNameIdentifierType"
1782 substitutionGroup="saml:NameIdentifier" />
1783 <xs:complexType name="EncryptableNameIdentifierType">
1784 <xs:simpleContent>
1785 <xs:extension base="saml:NameIdentifierType">
1786 <xs:attribute name="IssueInstant" type="xs:dateTime" />
1787 <xs:attribute name="Nonce" type="xs:string" use="optional" />
1788 </xs:extension>
1789 </xs:simpleContent>
1790 </xs:complexType>
1791 <xs:element name="EncryptedNameIdentifier" type="EncryptedNameIdentifierType" />
1792 <xs:complexType name="EncryptedNameIdentifierType">
1793 <xs:sequence>
1794 <xs:element ref="xenc:EncryptedData" />
1795 <xs:element ref="xenc:EncryptedKey" minOccurs="0" />
1796 </xs:sequence>
1797 </xs:complexType>
1798
1799 <xs:element name="AuthenticationStatement" type="AuthenticationStatementType" ↵
1800 substitutionGroup="saml:Statement" />
1801 <xs:complexType name="AuthenticationStatementType">
1802 <xs:complexContent>
1803 <xs:extension base="saml:AuthenticationStatementType">
1804 <xs:sequence>
1805 <xs:element name="AuthnContext" minOccurs="0">
1806 <xs:complexType>
1807 <xs:sequence>
1808 <xs:element name="AuthnContextClassRef" type="xs:anyURI" minOccurs="0" />
1809 <xs:choice>
1810 <xs:element ref="ac:AuthenticationContextStatement" />
1811 <xs:element name="AuthnContextStatementRef" type="xs:anyURI" />
1812 </xs:choice>
1813 </xs:sequence>
1814 </xs:complexType>
1815 </xs:element>
1816 </xs:sequence>
1817 <xs:attribute name="ReauthenticateOnOrAfter" type="xs:dateTime" use="optional" />
1818 <xs:attribute name="SessionIndex" type="xs:string" use="required" />
1819 </xs:extension>
1820 </xs:complexContent>
1821 </xs:complexType>
1822 <xs:element name="AuthnRequestEnvelope" type="AuthnRequestEnvelopeType" />
1823 <xs:complexType name="AuthnRequestEnvelopeType">
1824 <xs:complexContent>
1825 <xs:extension base="RequestEnvelopeType">
1826 <xs:sequence>
1827 <xs:element ref="AuthnRequest" />
1828 <xs:element ref="ProviderID" />
1829 <xs:element name="ProviderName" type="xs:string" minOccurs="0" />
1830 <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI" />
1831 <xs:element ref="IDPList" minOccurs="0" />
1832 <xs:element name="IsPassive" type="xs:boolean" minOccurs="0" />
1833 </xs:sequence>
1834 </xs:extension>
1835 </xs:complexContent>
1836 </xs:complexType>
1837 <xs:complexType name="RequestEnvelopeType">
1838 <xs:sequence>
1839 <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
1840 </xs:sequence>
1841 </xs:complexType>
1842 <xs:element name="IDPList" type="IDPListType" />
1843 <xs:complexType name="IDPListType">
1844 <xs:sequence>
1845 <xs:element ref="IDPEntries" />
1846 <xs:element ref="GetComplete" minOccurs="0" />

```

```

1847     </xs:sequence>
1848 </xs:complexType>
1849 <xs:element name="IDPEntry">
1850   <xs:complexType>
1851     <xs:sequence>
1852       <xs:element ref="ProviderID"/>
1853       <xs:element name="ProviderName" type="xs:string" minOccurs="0"/>
1854       <xs:element name="Loc" type="xs:anyURI"/>
1855     </xs:sequence>
1856   </xs:complexType>
1857 </xs:element>
1858 <xs:element name="IDPEntries">
1859   <xs:complexType>
1860     <xs:sequence>
1861       <xs:element ref="IDPEntry" maxOccurs="unbounded"/>
1862     </xs:sequence>
1863   </xs:complexType>
1864 </xs:element>
1865 <xs:element name="GetComplete" type="xs:anyURI"/>
1866 <xs:element name="AuthnResponseEnvelope" type="AuthnResponseEnvelopeType"/>
1867 <xs:complexType name="AuthnResponseEnvelopeType">
1868   <xs:complexContent>
1869     <xs:extension base="ResponseEnvelopeType">
1870       <xs:sequence>
1871         <xs:element ref="AuthnResponse"/>
1872         <xs:element name="AssertionConsumerServiceURL" type="xs:anyURI"/>
1873       </xs:sequence>
1874     </xs:extension>
1875   </xs:complexContent>
1876 </xs:complexType>
1877 <xs:complexType name="ResponseEnvelopeType">
1878   <xs:sequence>
1879     <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1880   </xs:sequence>
1881 </xs:complexType>
1882 <xs:element name="RegisterNameIdentifierRequest" type="RegisterNameIdentifierRequestType"/>
1883 <xs:complexType name="RegisterNameIdentifierRequestType">
1884   <xs:complexContent>
1885     <xs:extension base="samlp:RequestAbstractType">
1886       <xs:sequence>
1887         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1888         <xs:element ref="ProviderID"/>
1889         <xs:element ref="IDPProvidedNameIdentifier"/>
1890         <xs:element ref="SPPProvidedNameIdentifier" minOccurs="0"/>
1891         <xs:element ref="OldProvidedNameIdentifier"/>
1892         <xs:element ref="RelayState" minOccurs="0"/>
1893       </xs:sequence>
1894     </xs:extension>
1895   </xs:complexContent>
1896 </xs:complexType>
1897 <xs:element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1898 <xs:element name="SPPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1899 <xs:element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1900 <xs:element name="RegisterNameIdentifierResponse" type="StatusResponseType"/>
1901 <xs:complexType name="StatusResponseType">
1902   <xs:complexContent>
1903     <xs:extension base="samlp:ResponseAbstractType">
1904       <xs:sequence>
1905         <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1906         <xs:element ref="ProviderID"/>
1907         <xs:element ref="samlp:Status"/>
1908         <xs:element ref="RelayState" minOccurs="0"/>
1909       </xs:sequence>
1910     </xs:extension>
1911   </xs:complexContent>
1912 </xs:complexType>
1913 <xs:element name="FederationTerminationNotification

```

```

1914 " type="FederationTerminationNotificationType"/>
1915   <xs:complexType name="FederationTerminationNotificationType">
1916     <xs:complexContent>
1917       <xs:extension base="samlp:RequestAbstractType">
1918         <xs:sequence>
1919           <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1920           <xs:element ref="ProviderID"/>
1921           <xs:element ref="saml:NameIdentifier"/>
1922         </xs:sequence>
1923         <xs:attribute ref="consent" use="optional"/>
1924       </xs:extension>
1925     </xs:complexContent>
1926   </xs:complexType>
1927   <xs:element name="LogoutRequest" type="LogoutRequestType"/>
1928   <xs:complexType name="LogoutRequestType">
1929     <xs:complexContent>
1930       <xs:extension base="samlp:RequestAbstractType">
1931         <xs:sequence>
1932           <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1933           <xs:element ref="ProviderID"/>
1934           <xs:element ref="saml:NameIdentifier"/>
1935           <xs:element name="SessionIndex" type="xs:string" minOccurs="0"
1936 maxOccurs="unbounded"/>
1937           <xs:element ref="RelayState" minOccurs="0"/>
1938         </xs:sequence>
1939         <xs:attribute ref="consent" use="optional"/>
1940         <xs:attribute name="NotOnOrAfter" type="xs:dateTime" use="optional"/>
1941       </xs:extension>
1942     </xs:complexContent>
1943   </xs:complexType>
1944   <xs:element name="LogoutResponse" type="StatusResponseType"/>
1945   <xs:element name="NameIdentifierMappingRequest" type="NameIdentifierMappingRequestType"/>
1946   <xs:complexType name="NameIdentifierMappingRequestType">
1947     <xs:complexContent>
1948       <xs:extension base="samlp:RequestAbstractType">
1949         <xs:sequence>
1950           <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1951           <xs:element ref="ProviderID"/>
1952           <xs:element ref="saml:NameIdentifier"/>
1953           <xs:element name="TargetNamespace" type="md:entityIDType"/>
1954         </xs:sequence>
1955         <xs:attribute ref="consent" use="optional"/>
1956       </xs:extension>
1957     </xs:complexContent>
1958   </xs:complexType>
1959   <xs:element name="NameIdentifierMappingResponse" type="NameIdentifierMappingResponseType"/>
1960   <xs:complexType name="NameIdentifierMappingResponseType">
1961     <xs:complexContent>
1962       <xs:extension base="samlp:ResponseAbstractType">
1963         <xs:sequence>
1964           <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
1965           <xs:element ref="ProviderID"/>
1966           <xs:element ref="samlp:Status"/>
1967           <xs:element ref="saml:NameIdentifier" minOccurs="0"/>
1968         </xs:sequence>
1969       </xs:extension>
1970     </xs:complexContent>
1971   </xs:complexType>
1972 </xs:schema>
1973
1974

```

References

1975 Normative

- 1977 [LibertyBindProf] Cantor, Scott, Kemp, John, Champagne, Darryl, eds. "Liberty ID-FF Bindings and
1978 Profiles Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004).
1979 <http://www.projectliberty.org/specs>
- 1980 [LibertyGlossary] "Liberty Technical Glossary," Version 1.4, Liberty Alliance Project (14 Dec 2004).
1981 <http://www.projectliberty.org/specs> Hodges, Jeff, eds.
- 1982 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.1, Liberty
1983 Alliance Project (14 December 2004). <http://www.projectliberty.org/specs>
- 1984 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet
1985 Engineering Task Force (March 1997). <http://www.ietf.org/rfc/rfc2119.txt> [March 1997].
- 1986 [RFC3280] Housley, R., eds. (April 2002). "Internet X.509 Public Key Infrastructure Certificate and
1987 Certificate Revocation List (CRL) Profile," RFC 3280, The Internet Engineering Task Force
1988 <http://www.ietf.org/rfc/rfc3280.txt> [April 2002].
- 1989 [SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Assertions and Protocol
1990 for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification,
1991 version 1.1, Organization for the Advancement of Structured Information Standards [http://www.oasis-](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)
1992 [open.org/committees/documents.php?wg_abbrev=security](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)
- 1993 [SAMLBind11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Bindings and Profiles
1994 for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification,
1995 version 1.1, Organization for the Advancement of Structured Information Standards [http://www.oasis-](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)
1996 [open.org/committees/documents.php?wg_abbrev=security](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security)
- 1997 [Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May
1998 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium
1999 <http://www.w3.org/TR/xmlschema-1/>
- 2000 [Schema2] Biron, Paul V., Malhotra, Ashok, eds. (May 2002). "XML Schema Part 2: Datatypes," Recommendation,
2001 World Wide Web Consortium <http://www.w3.org/TR/xmlschema-2/>
- 2002 [XMLDsig] Eastlake, Donald, Reagle, Joseph, Solo, David, eds. (12 Feb 2002). "XML-Signature Syntax and
2003 Processing," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlsig-core>
- 2004 [XMLCanon] Boyer, John, Eastlake, Donald, Reagle, Joseph, eds. (18 July 2002). "Exclusive XML Canonicaliza-
2005 tion," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xml-exc-c14n>
- 2006 [xmllenc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing,"
2007 W3C Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmllenc-core/>

2008 Informative

- 2009 [LibertyImplGuide] "Liberty ID-FF Implementation Guidelines," Version 1.2, Liberty Alliance Project (18 April
2010 2004). <http://www.projectliberty.org/specs> Thompson, Peter, Champagne, Darryl, eds.