



1

2

Liberty ID-FF Protocols and Schema Specification

3

Version 1.2 – 08

4

11 April 2003

5

6

Editors:

7

Scott Cantor, Internet2 / The Ohio State University

8

John Kemp, IEEE-ISTO

9

Contributors:

10

Robert Aarts, Nokia

11

Xavier Serret, Gemplus

12

Greg Whitehead, Trustgenix

13

14

Abstract:

15

This specification defines a core set of protocols that collectively provide a solution for identity federation

16

management, cross-domain authentication, and session management. This specification contains the core protocols

17

and schema for Liberty identity federation. The reader is presumed to be generally familiar with the SAML Core

18

specification.

19

20

Copyright © 2003 Liberty Alliance Project

21

21 **Notice**

22 Copyright © 2002,2003 ActivCard; American Express Travel Related Services; America Online, Inc.; Bank of
23 America; Bell Canada; Catavault; Cingular Wireless; Cisco Systems, Inc.; Citigroup; Communicator, Inc.; Consignia;
24 Cyberun Corporation; Deloitte & Touche LLP; Earthlink, Inc.; Electronic Data Systems, Inc.; Entrust, Inc.; Ericsson;
25 Fidelity Investments; France Telecom; Gemplus; General Motors; Hewlett-Packard Company; i2 Technologies, Inc.;
26 Internet2; Intuit Inc.; MasterCard International; NEC Corporation; Netegrity; NeuStar; Nextel Communications;
27 Nippon Telegraph and Telephone Company; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.; OneName
28 Corporation; Openwave Systems Inc.; Phaos Technology; PricewaterhouseCoopers LLP; Register.com; RSA Security
29 Inc; Sabre Holdings Corporation; SAP AG; SchlumbergerSema; SK Telecom; Sony Corporation; Sun Microsystems,
30 Inc.; Trustgenix; United Airlines; VeriSign, Inc.; Visa International; Vodafone Group Plc; Wave Systems;. All rights
31 reserved.

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

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

45 Liberty Alliance Project
46 Licensing Administrator
47 c/o IEEE-ISTO
48 445 Hoes Lane
49 Piscataway, NJ 08855-1331, USA
50 info@projectliberty.org

51

51 **History**

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

52

52 **Table of Contents**

53 1 Introduction 5
54 1.1 Notation 5
55 1.2 Overview..... 6
56 2 Schema Declarations..... 6
57 2.1 Schema Header and Namespace Declarations..... 6
58 2.2 Type and Element Declarations..... 6
59 3 Protocols..... 7
60 3.1 General Requirements 7
61 3.1.1 XML Signature..... 7
62 3.1.2 Protocol and Assertion Versioning 7
63 3.1.3 Provider and Affiliation ID Uniqueness..... 7
64 3.1.4 Name Identifier Construction 8
65 3.1.5 Signature Verification 8
66 3.1.6 Security 8
67 3.1.7 Time Values 8
68 3.1.8 Time Synchronization 9
69 3.1.9 Response Status Codes..... 9
70 3.1.10 Use of <Extension> in Protocols..... 9
71 3.1.11 Interoperation with previous Liberty Implementations 9
72 3.1.12 Use of the consent attribute..... 10
73 3.2 Single Sign-On and Federation Protocol..... 10
74 3.2.1 Request..... 10
75 3.2.2 Response 14
76 3.2.3 Processing Rules..... 18
77 3.2.4 Request Envelope 23
78 3.2.5 Response Envelope 26
79 3.3 Name Registration Protocol 27
80 3.3.1 Request..... 27
81 3.3.2 Response 29
82 3.3.3 Processing Rules..... 30
83 3.4 Federation Termination Notification Protocol..... 30
84 3.4.1 Message..... 31
85 3.4.2 Processing Rules..... 32
86 3.5 Single Logout Protocol 32
87 3.5.1 Request..... 32
88 3.6 Introduction Notification Protocol..... 35
89 3.6.1 Message..... 36
90 3.6.2 Processing Rules..... 37
91 3.7 Provider Relationship Termination Protocol..... 37
92 3.7.1 Message..... 37
93 3.7.2 Response 38
94 3.7.3 Processing Rules..... 39
95 4 Schema Definition 39
96 5 References 45
97

97

98 1 Introduction

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

102 1.1 Notation

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

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

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

114 Listings of schemas appear like this.

115

116 Listings of instance fragments appear like this.

117

118 The following namespaces are referred to in this document:

119 The prefix `lib:` stands for the Liberty ID-FF namespace (`urn:liberty:iff:1.2`). This namespace is
120 the default for instance fragments, type names, and element names in this document.

121 The prefix `ac:` stands for the Liberty authentication context namespace (`urn:liberty:ac:1.2`)

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

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

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

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

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

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

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

137 Definitions for Liberty-specific terms can be found in [[LibertyGloss](#)].

138 1.2 Overview

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

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

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

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

153 2 Schema Declarations

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

156 2.1 Schema Header and Namespace Declarations

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

```
158 <?xml version="1.0" encoding="UTF-8"?>  
159 <schema targetNamespace="urn:liberty:iff:1.2" xmlns:lib="urn:liberty:iff:1.2"  
160     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
161     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"  
162     xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"  
163     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ac="urn:liberty:ac:1.2"  
164     xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"  
165     attributeFormDefault="unqualified">  
166     <import namespace="urn:oasis:names:tc:SAML:1.0:assertion" schemaLocation="  
167     http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-  
168     01.xsd"/>  
169     <import namespace="urn:oasis:names:tc:SAML:1.0:protocol" schemaLocation="  
170     http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-protocol-  
171     01.xsd"/>  
172     <import namespace="http://www.w3.org/2001/04/xmlenc#"  
173     schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd"/>  
174     <import namespace="http://www.w3.org/2000/09/xmldsig#"  
175     schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>  
176     <import namespace="urn:liberty:ac:1.2"  
177     schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-  
178     authentication-context-v1.2.xsd"/>  
179     <include schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-  
180     utility-v1.0.xsd"/>
```

181 2.2 Type and Element Declarations

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

```
183 <simpleType name="ProviderIDType">
184   <restriction base="anyURI" />
185 </simpleType>
186 <element name="ProviderID" type="lib:ProviderIDType" />
187 <element name="AffiliationID" type="lib:ProviderIDType" />
```

188 3 Protocols

189 The Liberty protocol suite consists of the following protocols:

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

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

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

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

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

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

202 3.1 General Requirements

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

204 3.1.1 XML Signature

205 The XML signature specification calls out a general XML syntax for signing data with many flexibilities and choices.
206 All signed XML entities MUST adhere to the “XML Signature Profile” constraints defined in [\[SAMLCore\]](#).

207 3.1.2 Protocol and Assertion Versioning

208 Version information appears in protocol messages and assertions defined in this specification. This specification
209 defines version 1.2 for the protocol messages and assertions. Version numbering of assertions is independent of the
210 version numbering of the protocol messages.

211 This specification follows the version numbering requirements, processing rules, and error conditions specified in
212 “SAML Versioning” in [\[SAMLCore\]](#).

213 3.1.3 Provider and Affiliation ID Uniqueness

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

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

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

221 3.1.4 Name Identifier Construction

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

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

230 3.1.5 Signature Verification

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

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

237 Enveloped Signature [\[XMLDsig\]](#)

238 Exclusive XML Canonicalization [\[XMLCanon\]](#)

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

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

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

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

252 3.1.6 Security

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

258 3.1.7 Time Values

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

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

265 **3.1.8 Time Synchronization**

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

267 The Identity Provider SHOULD NOT include a `NotBefore` attribute on the `Conditions` element of the assertion it
268 generates which contains the time the assertion was generated.

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

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

274 `<LogoutRequest>` is received

275 The user's session times out via normal means

276 The `ReauthenticateOnOrAfter` time on the `<AuthenticationStatement>` used to authenticate the
277 principal, if any, is reached

278 **3.1.9 Response Status Codes**

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

283 **3.1.10 Use of `<Extension>` in Protocols**

284 Most of the protocol messages defined in this document contain a generic `<Extension>` element that permits the
285 inclusion of arbitrary XML content representing agreements between providers that go beyond the bounds of the
286 specification. Implementers should understand that while extension content can be of a complex nature when fully
287 XML-capable profiles are used, this is not the case for profiles that bind protocol messages to a URL query string.

288 When using such profiles, the extension content MUST be deterministically expressible as a sequence of name/value
289 pairs. This requires that the XML content MUST be confined to attributes and simple element content in the "null"
290 namespace with non-overlapping local names. The total size of extension content SHOULD be minimized.

291 **3.1.11 Interoperation with previous Liberty Implementations**

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

- 296 • Metadata SHOULD be used to the greatest extent possible to identify the capabilities of a provider, so that
297 the proper messages can be sent. See [\[LibMetadata\]](#).
- 298 • Version 1.2 implementations SHOULD avoid sending 1.2 requests or notifications to providers that only
299 implement earlier versions of the specification.
- 300 • When in doubt, 1.2 implementations SHOULD send 1.1 requests and notifications when the older
301 specification meets the requirements of the transaction.

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

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

305 **3.1.12 Use of the consent attribute**

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

308 Three values are defined for this attribute:

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

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

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

315 **3.2 Single Sign-On and Federation Protocol**

316 The Single Sign-On and Federation Protocol defines a request and response protocol by which single sign-on and
317 identity federation occurs. The protocol works as follows:

- 318 1. A service provider issues an <AuthnRequest> request to an identity provider, instructing the
319 identity provider to provide an authentication assertion to the service provider. Optionally, the service
320 provider MAY request that the identity be federated.
- 321 2. The identity provider responds with either an <AuthnResponse> containing authentication
322 assertions to the service provider or an artifact that can be de-referenced into an authentication
323 assertion. Additionally, the identity provider potentially federates the Principal's identity at the
324 identity provider with the Principal's identity at the service provider.

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

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

332 **3.2.1 Request**

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

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

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

338 Federate the Principal's identity at the identity provider with the Principal's identity at the service provider.

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

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

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

343 Restrict or limit the identity provider's ability to proxy the authentication request to additional identity
344 providers.

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

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

349 **3.2.1.1 Element <AuthnRequest>**

350 The <AuthnRequest> is defined as an extension of **samlp:RequestAbstractType**. The RequestID attribute in
351 **samlp:RequestAbstractType** has uniqueness requirements placed on it by [\[SAMLCore\]](#), which require it to have the
352 properties of a nonce.

353 The elements of the request are as follows:

354 **Extension** [Optional]

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

358 **ProviderID** [Required]

359 The service provider's URI-based identifier.

360 **AffiliationId** [Optional]

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

362 **NameIDPolicy** [Optional]

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

368 **IsPassive** [Optional]

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

372 **ForceAuthn** [Optional]

373 Controls whether the identity provider authenticates the Principal regardless of whether the Principal is already
374 authenticated. This element is specified only when <IsPassive> is "false." If <ForceAuthn> is "true,"
375 specifies that the identity provider MUST always authenticate the Principal, regardless of whether the Principal is
376 presently authenticated. If "false," specifies that the identity provider MUST re-authenticate the user only if the
377 Principal is not presently authenticated. If not specified, "false" is presumed.

378 **ProtocolProfile** [Optional]

379 The protocol profile that the service provider wishes to use for the response. If the element is not specified, the
380 default protocol profile is <http://projectliberty.org/profiles/brws-art>, defined in
381 [\[LibertyBindProf\]](#).

382 **AssertionConsumerServiceID** [Optional]

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

385 AuthnContext [Optional]

386 Information describing which authentication context the service provider desires the identity provider to use in
387 authenticating the Principal.

388 RelayState [Optional]

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

392 id [Optional]

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

395 AuthnContextComparison [Optional]

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

400 ProxyCount [Optional]

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

403 IntroductionArtifact [Optional]

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

406 consent [Optional]

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

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

409 AuthnContextClassRef [Optional]

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

412 AuthnContextStatementRef [Optional]

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

415

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

```
417 <element name="AuthnRequest" type="lib:AuthnRequestType" />
418 <complexType name="AuthnRequestType">
419   <complexContent>
420     <extension base="samlp:RequestAbstractType">
421       <sequence>
422         <element ref="lib:Extension minOccurs="0" maxOccurs="unbounded" />
423         <element ref="lib:ProviderID" />
424         <element ref="lib:AffiliationId" minOccurs="0" />
425         <element ref="lib:NameIDPolicy" minOccurs="0" />
426         <element name="ForceAuthn" type="boolean" minOccurs="0" />
427         <element name="IsPassive" type="boolean" minOccurs="0" />
428         <element ref="lib:ProtocolProfile" minOccurs="0" />
429         <element name="AssertionConsumerServiceID" type="string" minOccurs="0" />

```

```

430     <element ref="lib:AuthnContext" minOccurs="0" />
431     <element ref="lib:RelayState" minOccurs="0" />
432     <element name="AuthnContextComparison"
433     type="lib:AuthnContextComparisonType" minOccurs="0" />
434     <element name="ProxyCount" type="nonNegativeInteger" minOccurs="0" />
435     <element ref="lib:IntroductionArtifact" minOccurs="0" />
436   </sequence>
437   <attribute name="id" type="ID" use="optional" />
438   <attribute ref="lib:consent" use="optional" />
439 </extension>
440 </complexContent>
441 </complexType>
442 <simpleType name="NameIDPolicyType">
443   <restriction base="string">
444     <enumeration value="none" />
445     <enumeration value="temporary" />
446     <enumeration value="federated" />
447     <enumeration value="any" />
448   </restriction>
449 </simpleType>
450 <simpleType name="AuthnContextComparisonType">
451   <restriction base="string">
452     <enumeration value="exact" />
453     <enumeration value="minimum" />
454     <enumeration value="better" />
455   </restriction>
456 </simpleType>
457 <element name="NameIDPolicy" type="lib:NameIDPolicyType" />
458 <element name="RelayState" type="string" />
459 <element name="ProtocolProfile" type="anyURI" />
460 <element name="AuthnContext">
461   <complexType>
462     <choice>
463       <element name="AuthnContextClassRef" type="anyURI" maxOccurs="unbounded" />
464       <element name="AuthnContextStatementRef" type="anyURI "
465       maxOccurs="unbounded" />
466     </choice>
467   </complexType>
468 </element>
469 <element name="IntroductionArtifact" type="string" />

```

3.2.1.2 Example

```

471 <lib:AuthnRequest id="12345" RequestID="RPCUk211+GVz+t11LURp51oFvJXk"
472   MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"
473   IssueInstant="2001-12-17T21:42:4Z" xmlns:lib="urn:liberty:iff:1.2">
474   <ds:Signature> ... </ds:Signature>
475   <lib:ProviderID>http://ServiceProvider.com</lib:ProviderID>
476   <lib:NameIDPolicy>federate</lib:NameIDPolicy>
477   <lib:ForceAuthn>>false</lib:ForceAuthn>
478   <lib:IsPassive>>false</lib:IsPassive>
479   <lib:ProtocolProfile>http://projectliberty.org/profiles/brws-
480   post</lib:ProtocolProfile>
481   <lib:AuthnContext>
482
483     <lib:AuthnContextClassRef>http://projectliberty.org/schemas/authctx/clo
484     sses/Password-ProtectedTransport</lib:AuthnContextClassRef>
485   </lib:AuthnContext>
486   <lib:RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</lib:RelayState>
487   <lib:AuthnContextComparison>exact</lib:AuthnContextComparison>
488   <lib:ProxyCount>1</lib:ProxyCount>
489 </lib:AuthnRequest>

```

490 3.2.2 Response

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

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

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

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

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

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

509 If the <NameIDPolicy> element is "any", then evaluation proceeds as if the value were "federated". If the policy
510 for the Principal forbids federation, then evaluation MAY proceed as if the value were "temporary".

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

512 Identity providers MUST include a <saml:AudienceRestrictionCondition> element that specifies the
513 intended consumers of the assertion. The <saml:Audience> element MUST be set to the intended recipient's
514 ProviderID. The recipient MUST validate that it is the intended viewer before using the assertion.

515 Identity providers MAY include a SessionIndex attribute in resulting authentication statements, which is used to
516 aid the identity provider in managing multiple sessions with the Principal. If the identity provider includes this
517 SessionIndex attribute, subsequent messages from the service provider to the identity provider that are session-
518 dependent MUST include this SessionIndex attribute.

519 Identity providers MAY include other types of statements in the assertion(s) returned, depending on agreements
520 between providers and other specifications that provide additional functionality.

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

524 3.2.2.1 Element <AuthnResponse>

525 The type AuthnResponseType is extended from samlp:ResponseType.

526 The response contains the following elements:

527 Extension [Optional]

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

529 ProviderID [Required]

530 The identity provider's URI-based identifier.

531 RelayState [Optional]

532 This contains state information being relayed.

533 id [Optional]

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

536 consent [Optional]

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

538 The schema fragment is as follows:

```
539 <element name="AuthnResponse" type="lib:AuthnResponseType"/>
540 <complexType name="AuthnResponseType">
541   <complexContent>
542     <extension base="saml:ResponseType">
543       <sequence>
544         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
545         <element ref="lib:ProviderID"/>
546         <element ref="lib:RelayState" minOccurs="0"/>
547       </sequence>
548       <attribute name="id" type="ID" use="optional"/>
549       <attribute ref="lib:consent" use="optional"/>
550     </extension>
551   </complexContent>
552 </complexType>
```

553 3.2.2.2 Element <AssertionType>

554 Authentication assertions provided in an <AuthnResponse> element MUST be of type **AssertionType**, which is an
555 extension of **saml:AssertionType**, so that the RequestID attribute from the original <AuthnRequest> is
556 included in the InResponseTo attribute in the <Assertion> element. This is done because it is not required that
557 the <AuthnResponse> element itself be signed. Instead, the individual <Assertion> elements contained must
558 each be signed. The id attribute is also included to facilitate such signatures (see section 3.1.5, Signature Verification).

559 The schema fragment is as follows:

```
560 <complexType name="AssertionType">
561   <complexContent>
562     <extension base="saml:AssertionType">
563       <attribute name="InResponseTo" type="saml:IDReferenceType"/>
564       <attribute name="id" type="ID" use="optional"/>
565     </extension>
566   </complexContent>
567 </complexType>
```

568 3.2.2.3 Type SubjectType

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

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

- 572
- 573 • Identifiers communicated on behalf of Principals that have federated their identity MUST contain a
Format URI of urn:liberty:iff:nameid:federated.

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

577 The schema fragment is as follows:

```
578 <complexType name="SubjectType">  
579   <complexContent>  
580     <extension base="saml:SubjectType">  
581       <sequence>  
582         <element ref="lib:IDPProvidedNameIdentifier"/>  
583       </sequence>  
584     </extension>  
585   </complexContent>  
586 </complexType>
```

587 3.2.2.4 Type EncryptedSubjectType

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

593 EncryptedNameIdentifier [Required]

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

595 xenc:EncryptedKey [Optional]

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

597 The schema fragment is as follows:

```
598 <complexType name="EncryptedSubjectType">  
599   <complexContent>  
600     <extension base="saml:SubjectType">  
601       <sequence>  
602         <element name="EncryptedNameIdentifier"  
603         type="xenc:EncryptedDataType"/>  
604         <element ref="xenc:EncryptedKey" minOccurs="0"/>  
605       </sequence>  
606     </extension>  
607   </complexContent>  
608 </complexType>
```

609 3.2.2.5 Type AuthenticationStatementType

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

612 AuthnContext [Optional]

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

616 ReauthenticateOnOrAfter [Optional]

617 The time at, or after which the service provider reauthenticates the Principal with the identity provider (as required
618 in [Section 3.2](#) above).

619 SessionIndex [Optional]

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

623 When an <AuthnContext> element is specified, the **saml:AuthenticationMethod** attribute on the
624 <saml:AuthenticationStatement> MUST be "http://projectliberty.org/schemas/authctx/2002/05".

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

629 The schema fragment is as follows:

```
630 <complexType name="AuthenticationStatementType">
631   <complexContent>
632     <extension base="saml:AuthenticationStatementType">
633       <sequence>
634         <element name="AuthnContext" minOccurs="0">
635           <complexType>
636             <sequence>
637               <element name="AuthnContextClassRef" type="anyURI" minOccurs="0"/>
638               <choice>
639                 <element ref="ac:AuthenticationContextStatement"/>
640                 <element name="AuthnContextStatementRef" type="anyURI"/>
641               </choice>
642             </sequence>
643           </complexType>
644         </element>
645       </sequence>
646       <attribute name="ReauthenticateOnOrAfter" type="dateTime" use="optional"/>
647       <attribute name="SessionIndex" type="string" use="optional"/>
648     </extension>
649   </complexContent>
650 </complexType>
```

651 3.2.2.6 Example

```
652 <lib:AuthnResponse id="54321" ResponseID="9hhuujalbc744hGJn5Q9A5yvEIgS"
653   InResponseTo="Zon3WjJ2KL7j+bJu7MuIr4Pt2go5" MajorVersion="1"
654   MinorVersion="2" consent="urn:liberty:consent:obtained"
655   IssueInstant="2002-10-31T21:55:41Z">
656   <samlp:Status>
657     <samlp:StatusCode Value="samlp:Success"/>
658   </samlp:Status>
659   <saml:Assertion id="12345" MajorVersion="1" MinorVersion="0"
660     AssertionID="e06e5a28-bc80-4ba6-9ecb-712949db686e"
661     Issuer="http://IdentityProvider.com" IssueInstant="2001-12-17T09:30:47Z"
662     InResponseTo="4e7c3772-4fa4-4a0f-99e8-7d719ff6067c"
663     xsi:type="AssertionType">
664     <saml:Conditions NotBefore="2001-12-17T09:30:47Z" NotOnOrAfter="2001-12-
665       17T09:35:47Z">
666       <saml:AudienceRestrictionCondition>
667         <saml:Audience>http://ServiceProvider.com</saml:Audience>
668       </saml:AudienceRestrictionCondition>
669     </saml:Conditions>
670     <saml:AuthenticationStatement AuthenticationInstant="2001-12-17T09:30:47Z"
671       SessionIndex="3" ReauthenticateOnOrAfter="2001-12-17T11:30:47Z"
672       xsi:type="AuthenticationStatementType"
673       AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
674     <saml:Subject xsi:type="SubjectType">
675       <saml:NameIdentifier
676         Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be35-
677         cc9e7db39e2b</saml:NameIdentifier>
```

```
678         <IDPProvidedNameIdentifier
679           Format="urn:liberty:iff:nameid:federated">342ad3d8-93ee-4c68-be35-
680           cc9e7db39e2b</IDPProvidedNameIdentifier>
681         </saml:Subject>
682       </saml:AuthenticationStatement>
683       <ds:Signature>...</ds:Signature>
684     </saml:Assertion>
685     <lib:ProviderID>http://IdentityProvider.com</lib:ProviderID>
686     <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
687   </AuthnResponse>
```

688 3.2.3 Processing Rules

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

691 The <ProviderID> in the request MAY be the Provider ID of a known provider with which the identity
692 provider has established a relationship, or an <IntroductionArtifact> element MAY be provided
693 to allow the identity provider to obtain evidence to establish such a relationship, possibly only
694 temporarily.

695 The <ProviderID> MUST be resolvable to at least one (default) assertion consumer service URL at the
696 requesting provider that the identity provider may use when returning the corresponding assertion
697 reference. The following rules apply to choosing the appropriate URL to use:

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

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

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

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

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

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

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

720 Failure to authenticate the Principal is indicated by a status code other than "samlp:Success." For failures,
721 assertions that contain any statement of a type other than **lib:IntroductionStatementType** MUST NOT
722 appear in the <AuthnResponse>.

723 <AffiliationId>, if present, MUST be the Affiliation ID of a known affiliation with which the identity
724 provider has an established relationship, and of which the requesting provider is a member. If present,
725 identity providers MUST establish and resolve federations based on the specified affiliation, not the
726 requesting provider. In addition, identity providers MAY retrieve information regarding the other
727 members of the affiliation group by querying metadata (see [LibMetadata]) and present a list of members
728 of <AffiliationId> to the Principal.

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

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

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

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

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

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

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

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

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

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

755 The `InResponseTo` attribute in all generated `<Assertion>` elements in the `<AuthnResponse>`
756 element MUST be set to the value of the `RequestID` attribute in the corresponding `<AuthnRequest>`
757 element.

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

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

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

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

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

775 If one or more `<AuthnContextStatementRef>` elements are included, then the resulting authentication
776 statement in the assertion (if any) MUST follow the rule specified in the

777 <AuthnContextComparison> element. If this requirement cannot be satisfied, the identity
778 provider MUST not include an authentication statement in the resulting assertion.

779 If the identity provider wishes to rely on a second identity provider as the source of the Principal's
780 authentication, then the following request processing rules apply. See also section 3.2.3.2 for additional
781 detail.

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

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

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

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

798 3.2.3.1 Introduction of Identity Providers

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

806 3.2.3.1.1 Type IntroductionStatementType

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

809 **ProviderID** or **AffiliationID** [One or more]

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

812 **Notification** [Optional]

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

816 The schema fragment is as follows:

```
817 <complexType name="IntroductionStatementType">  
818   <complexContent>  
819     <extension base="saml:SubjectStatementType">  
820       <choice maxOccurs="unbounded">  
821         <element ref="lib:ProviderID"/>  
822         <element ref="lib:AffiliationID"/>  
823       </choice>  
824       <element name="Notification" type="boolean" minOccurs="0"/>  
825     </extension>  
826   </complexContent>  
827 </complexType>
```

```
825     </extension>  
826     </complexContent>  
827 </complexType>
```

828 3.2.3.1.2 Processing Rules

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

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

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

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

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

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

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

852 If the identity provider federates the Principal's identity as a result of an introduction statement that contains
853 a <Notification> element with a value of "true", then it MUST use the protocol described in section
854 3.6 to notify the introducing identity provider of the federation event.

855 3.2.3.2 Dynamic Proxying of Identity Providers

856 An identity provider that is asked to authenticate a known Principal that it believes has already authenticated to
857 another identity provider MAY make an authentication request on behalf of the requesting provider to that
858 authenticating identity provider, provided the value in the <ProxyCount> element is greater than zero, if one exists.
859 The limit of such proxying steps is governed only by the <ProxyCount> on the request and local policy. When
860 creating the new authentication request:

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

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

866 If the request contained a <ProxyCount> element with a value greater than 0, then the new request MUST
867 contain a <ProxyCount> element with a value of one less than the original value. If the original request
868 does not contain a <ProxyCount> element, then the new request SHOULD contain a <ProxyCount>
869 element.

870 The authentication request and response are processed in normal fashion, in accordance with the rules given in section
871 3.2.3. Once the Principal has authenticated to the proxying identity provider, the following steps are followed:

872 The proxying identity provider prepares a new authentication assertion on its own behalf by copying in the
873 relevant information from the original assertion. The original assertion will be restricted by
874 AudienceRestrictionCondition to the identity provider, while the new assertion's condition will reference
875 the original requesting provider.

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

878 If the <NameIdentifier> is not anonymous, then the identity provider MUST include the Principal's
879 federated <IDPProvidedNameIdentifier> for the requesting provider or affiliation group, as well
880 as the <NameIdentifier> provided by the requesting provider or affiliation group, if any.

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

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

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

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

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

898 **3.2.3.3 Active Intermediaries**

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

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

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

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

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

915 **3.2.3.3.1 Processing Rules for Active Intermediaries**

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

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

- 919 • The intermediary MUST remove the enveloping <AuthnRequestEnvelope> before forwarding
920 the <AuthnRequest> element to the identity provider.

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

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

- 932 • The intermediary MUST remove the enveloping <AuthnResponseEnvelope> before forwarding
933 the <AuthnResponse> element to the service provider.
- 934 • The intermediary MUST send <AuthnResponse> messages received from the identity provider to
935 the service provider using the <AssertionConsumerServiceURL> contained within the
936 <AuthnResponseEnvelope> sent by the identity provider.

937 3.2.3.4 Status Code Values for Error Conditions

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

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

945 `lib:UnknownPrincipal`: Used by an identity provider to indicate that the Principal is not known to
946 it. MAY be accompanied by an introduction assertion.

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

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

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

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

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

958 3.2.4 Request Envelope

959 Some profiles MAY wrap the <AuthnRequest> element in an envelope. This envelope allows for extra processing
960 by an intermediary between the service provider and the identity provider. An example of an intermediary is a user

961 agent or proxy. Processing rules are given in section 3.2.3.1.1. Note that the envelope is for consumption by the
962 intermediary and is removed before the enveloped <AuthnRequest> element is forwarded to the identity provider.

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

966 **3.2.4.1 Element <AuthnRequestEnvelope>**

967 The authentication request envelope contains the following elements:

968 **Extension** [Optional]

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

972 **AuthnRequest** [Required]

973 The authentication request contained within the envelope.

974 **ProviderID** [Required]

975 The requestor's ProviderID.

976 **ProviderName** [Optional]

977 The human-readable name of the requestor.

978 **AssertionConsumerServiceURL** [Required]

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

981 **IDPList** [Optional]

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

983 **IsPassive** [Optional]

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

986 The schema fragment is as follows:

```
987 <element name="AuthnRequestEnvelope" type="lib:AuthnRequestEnvelopeType" />
988 <complexType name="AuthnRequestEnvelopeType">
989   <complexContent>
990     <extension base="lib:RequestEnvelopeType">
991       <sequence>
992         <element ref="lib:AuthnRequest" />
993         <element ref="lib:ProviderID" />
994         <element name="ProviderName" type="string" minOccurs="0" />
995         <element name="AssertionConsumerServiceURL" type="anyURI" />
996         <element ref="lib:IDPList" minOccurs="0" />
997         <element name="IsPassive" type="boolean" minOccurs="0" />
998       </sequence>
999     </extension>
1000   </complexContent>
1001 </complexType>
1002 <complexType name="RequestEnvelopeType">
1003   <sequence>
1004     <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1005   </sequence>
```

1006 `</complexType>`

1007 3.2.4.2 Element <IDPList>

1008 In the request envelope, some profiles may wish to allow the service provider to transport a list of identity providers to
1009 the user agent. This specification provides a schema that profiles SHOULD use for this purpose. The elements are as
1010 follows:

1011 IDPList

1012 The container element for an IDP List.

1013 IDPEntries

1014 Contains a list of identity provider entries.

1015 IDPEntry

1016 Describes an identity provider that the service provider supports.

1017 ProviderID

1018 The identity provider's ProviderID.

1019 ProviderName

1020 The identity provider's human-readable name.

1021 Loc

1022 The identity provider's URI, to which authentication requests may be sent. This SHOULD be set to the value of
1023 the identity provider's <SingleSignOnService> metadata element.

1024 GetComplete

1025 If the identity provider list is not complete, this element is included with a URI that points to where the complete
1026 list can be retrieved.

1027 The schema fragment is as follows:

```
1028 <element name="IDPList" type="lib:IDPListType"/>
1029 <complexType name="IDPListType">
1030   <sequence>
1031     <element ref="lib:IDPEntries"/>
1032     <element ref="lib:GetComplete" minOccurs="0" maxOccurs="unbounded"/>
1033   </sequence>
1034 </complexType>
1035 <element name="IDPEntry">
1036   <complexType>
1037     <sequence>
1038       <element ref="lib:ProviderID"/>
1039       <element name="ProviderName" type="string" minOccurs="0"/>
1040       <element name="Loc" type="anyURI"/>
1041     </sequence>
1042   </complexType>
1043 </element>
1044 <element name="IDPEntries">
1045   <complexType>
1046     <sequence>
1047       <element ref="lib:IDPEntry" maxOccurs="unbounded"/>
1048     </sequence>
1049   </complexType>
1050 </element>
1051 <element name="GetComplete" type="anyURI"/>
```

1052 3.2.4.3 Example

```

1053 <AuthnRequestEnvelope>
1054 <AuthnRequest> ... </AuthnRequest>
1055 <ProviderID>http://ServiceProvider.com</ProviderID>
1056 <ProviderName>Service Provider X</ProviderName>
1057 <AssertionConsumerServiceURL>http://ServiceProvider.com/lecp_assertion_
1058 consumer</AssertionConsumerServiceURL>
1059 <IDPList>
1060 <IDPEntries>
1061 <IDPEntry>
1062 <ProviderID>http://IdentityProvider.com</ProviderID>
1063 <ProviderName>Identity Provider X</ProviderName>
1064 <Loc>http://www.IdentityProvider.com/liberty/sso</Loc>
1065 </IDPEntry>
1066 </IDPEntries>
1067 <GetComplete>https://ServiceProvider.com/idplist?id=604bel136-fe91-441e-afb8-
1068 f88748ae3b8b </GetComplete>
1069 </IDPList>
1070 <IsPassive>0</IsPassive>
1071 </AuthnRequestEnvelope>

```

1072 3.2.5 Response Envelope

1073 As with the <AuthnRequest> element, some profiles MAY wrap the <AuthnResponse> element in an
1074 envelope. This envelope allows for extra processing by an intermediary (such as a user agent or proxy) between the
1075 identity provider and the service provider. Applicable processing rules are given in section 3.2.3.1.1. Note that the
1076 envelope is for consumption by the intermediary and is removed prior to the forwarding of the enveloped
1077 <AuthnResponse> element to the service provider.

1078 3.2.5.1 Element <AuthnResponseEnvelope>

1079 The authentication response envelope contains the following elements:

1080 Extension [Optional]

1081 Optional container for protocol extensions established by agreement between providers. Implementors should note
1082 that this element may not contain content from the core Liberty namespace (which is prevented at the schema level
1083 by requiring namespace="##other").

1084 AuthnResponse [Required]

1085 The enveloped authentication response.

1086 AssertionConsumerServiceURL [Required]

1087 The service provider's URL where the authentication response should be sent. This element's value SHOULD be
1088 obtained from the element of the same name in the service provider's Provider Metadata.

1089 The schema fragment is as follows:

```

1090 <element name="AuthnResponseEnvelope" type="lib:AuthnResponseEnvelopeType" />
1091 <complexType name="AuthnResponseEnvelopeType">
1092 <complexContent>
1093 <extension base="lib:ResponseEnvelopeType">
1094 <sequence>
1095 <element ref="lib:AuthnResponse" />
1096 <element name="AssertionConsumerServiceURL" type="anyURI" />
1097 </sequence>
1098 </extension>
1099 </complexContent>
1100 </complexType>
1101 <complexType name="ResponseEnvelopeType">
1102 <sequence>

```

```
1103     <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1104     </sequence>
1105 </complexType>
```

1106 3.2.5.2 Example

```
1107 <AuthnResponseEnvelope>
1108   <AuthnResponse> ... </AuthnResponse>
1109 <AssertionConsumerServiceURL>
1110   http://ServiceProvider.com/lecp_assertion_consumer
1111 </AssertionConsumerServiceURL>
1112 </AuthnResponseEnvelope>
```

1113 3.3 Name Registration Protocol

1114 During federation, the identity provider generates an opaque handle that serves as the initial name identifier that both
1115 the service provider and the identity provider use in referring to the Principal when communicating with each other.
1116 This name identifier is termed the <IDPProvidedNameIdentifier>.

1117 Subsequent to federation, the service provider MAY register a different opaque handle with the identity provider. This
1118 opaque handle is termed the <SPPProvidedNameIdentifier>. Until the service provider registers a different
1119 name, the identity provider will use <IDPProvidedNameIdentifier> to refer to the Principal when
1120 communicating with the service provider.

1121 After a service provider's name registration, the identity provider MUST use the
1122 <SPPProvidedNameIdentifier> for <saml:NameIdentifier> elements when communicating to the
1123 service provider about the Principal. The service provider MUST use the current (most recently supplied)
1124 <IDPProvidedNameIdentifier> for <saml:NameIdentifier> elements when communicating to the
1125 identity provider about the Principal.

1126 Either the service provider or the identity provider MAY register a new name identifier for a Principal with each other
1127 at any time following federation. The name identifiers specified by providers SHOULD adhere to the following
1128 guidelines:

- 1129 • The name identifier SHOULD be unique across the identity providers with which the Principal's identity is
1130 federated.
- 1131 • The name identifier SHOULD be unique within the group of name identifiers that have been registered
1132 with the identity provider by this service provider.

1133 3.3.1 Request

1134 To register a <SPPProvidedNameIdentifier> with an identity provider, the service provider sends a
1135 <RegisterNameIdentifierRequest> message.

1136 The same <RegisterNameIdentifierRequest> message may be sent by an identity provider, seeking to
1137 change the <IDPProvidedNameIdentifier> stored by the service provider.

1138 The <RegisterNameIdentifierRequest> message SHOULD be signed.

1139 3.3.1.1 Element <RegisterNameIdentifierRequest>

1140 The elements of the message are as follows:

1141 Extension [Optional]

1142 Optional container for protocol extensions established by agreement between providers.

1143 ProviderID [Required]

1144 The provider's identifier.

1145 IDPProvidedNameIdentifier [Required]

1146 The name identifier the service provider should use when communicating with the identity provider.

1147 SPPProvidedNameIdentifier [Required]

1148 The name identifier the identity provider should use when communicating to the service provider.

1149 OldProvidedNameIdentifier [Required]

1150 In the case of either provider choosing to request a change of provided name identifiers, this element holds the
1151 previous version. For a service provider making their first name change following federation, the
1152 <OldProvidedNameIdentifier> will contain the current <IDPProvidedNameIdentifier>. The
1153 <SPPProvidedNameIdentifier> will contain the new name that the service provider wishes the identity
1154 provider to use.

1155 id [Optional]

1156 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1157 RelayState [Optional]

1158 This contains state information that will be relayed back in the response. This data SHOULD be integrity-protected
1159 by the request author and MAY have other protections placed on it by the request author. An example of such
1160 protection is confidentiality.

1161 AffiliationId [Optional]

1162 If present and the requester is a service provider, indicates that the requesting service provider is acting as a
1163 member of the specified affiliation. If present and the requester is an identity provider, indicates that the receiving
1164 service provider should interpret the message as being from a member of the specified affiliation, and representing
1165 the group, rather than the individual provider.

1166 The schema fragment is as follows:

```
1167 <element name="RegisterNameIdentifierRequest"  
1168   type="lib:RegisterNameIdentifierRequestType" />  
1169 <complexType name="RegisterNameIdentifierRequestType">  
1170   <complexContent>  
1171     <extension base="samlp:RequestAbstractType">  
1172       <sequence>  
1173         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />  
1174         <element ref="lib:ProviderID" />  
1175         <element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType" />  
1176         <element name="SPPProvidedNameIdentifier" type="saml:NameIdentifierType" />  
1177         <element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType" />  
1178         <element ref="lib:RelayState" minOccurs="0" />  
1179         <element ref="lib:AffiliationId" minOccurs="0" />  
1180       </sequence>  
1181       <attribute name="id" type="ID" use="optional" />  
1182     </extension>  
1183   </complexContent>  
1184 </complexType>  
1185 <element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType" />  
1186 <element name="SPPProvidedNameIdentifier" type="saml:NameIdentifierType" />  
1187 <element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType" />  
1188
```

1189 3.3.1.2 Example

```
1190 <RegisterNameIdentifierRequest id="12345" RequestID="eb20e77f-d982-44f9-936e-  
1191   dd135bf437d4" MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-  
1192   17T09:30:47Z">
```

```
1193     <ds:Signature>...</ds:Signature>
1194     <ProviderID>http://ServiceProvider.com</ProviderID>
1195     <IDPProvidedNameIdentifier>342ad3d8-93ee-4c68-be35-
1196         cc9e7db39e2b</IDPProvidedNameIdentifier>
1197     <SPPProvidedNameIdentifier>e958019a</SPPProvidedNameIdentifier>
1198         <OldProvidedNameIdentifier>e895014a</OldProvidedNameIdentifier>
1199     <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1200 </RegisterNameIdentifierRequest>
```

1201 3.3.2 Response

1202 The responding provider MUST respond with <RegisterNameIdentifierResponse>, which is of type
1203 **StatusResponseType**. **StatusResponseType** is an extension of **samlp:ResponseType** and a <samlp:Status>
1204 element and a <RelayState> may exist in the body.

1205 This message SHOULD be signed.

1206 3.3.2.1 Element <RegisterNameIdentifierResponse>

1207 The elements of the message are as follows:

1208 Extension [Optional]

1209 Optional container for protocol extensions established by agreement between providers.

1210 ProviderID [Required]

1211 The provider's unique identifier.

1212 Status [Required]

1213 The status of the request processing.

1214 id [Optional]

1215 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1216 RelayState [Optional]

1217 This element contains state information that will be relayed back in the response, if it has been supplied in the
1218 request.

1219 The schema fragment is as follows:

```
1220 <element name="RegisterNameIdentifierResponse" type="lib:StatusResponseType"/>
1221 <complexType name="StatusResponseType">
1222   <complexContent>
1223     <extension base="samlp:ResponseAbstractType">
1224       <sequence>
1225         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1226         <element ref="lib:ProviderID"/>
1227         <element ref="samlp:Status"/>
1228         <element ref="lib:RelayState" minOccurs="0"/>
1229       </sequence>
1230       <attribute name="id" type="ID" use="optional"/>
1231     </extension>
1232   </complexContent>
1233 </complexType>
```

1234 **3.3.2.2 Example**

```
1235 <RegisterNameIdentifierResponse id="12345" ResponseID="74ffec0f-1165-4fa3-b088-  
1236 3dd2c2388b91" InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4"  
1237 MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"  
1238 Recipient="http://ServiceProvider.com">  
1239 <ds:Signature>...</ds:Signature>  
1240 <ProviderID>http://ServiceProvider.com</ProviderID>  
1241 <samlp:Status>  
1242 <samlp:StatusCode Value="samlp:Success"/>  
1243 </samlp:Status>  
1244 <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>  
1245 </RegisterNameIdentifierResponse>
```

1246 **3.3.3 Processing Rules**

1247 The recipient **MUST** validate any signature present on the message. To be considered valid, the signature
1248 provided **MUST** be the signature of the <ProviderID> contained in the message.

1249 If the request includes an <IDPProvidedNameIdentifier> for which no federation exists between the
1250 service provider and the identity provider, the provider **MUST** respond with a <samlp:Status> element
1251 containing a second-level <samlp:StatusCode> of lib:FederationDoesNotExist. Otherwise,
1252 the identity provider **MUST** use <SPPProvidedNameIdentifier> when subsequently communicating to
1253 the service provider regarding this Principal.

1254 Either provider **MAY** choose to change their provided name identifier. In this case, the
1255 <OldProvidedNameIdentifier> should contain the previous version of their name identifier. When
1256 a service provider chooses to change their provided name identifier, the
1257 <OldProvidedNameIdentifier> should contain the current <SPPProvidedNameIdentifier>.
1258 Note that when they first change their name, this will be equal to the
1259 <IDPProvidedNameIdentifier>. Similarly, when an identity provider wishes to change their
1260 provided name identifier, they will move the previous version to the <OldProvidedNameIdentifier>
1261 when sending this message.

1262 Changes to these identifiers may take a potentially significant amount of time to propagate through the systems at
1263 both the sender and the receiver. Implementations **MAY** wish to allow each party to accept either identifier
1264 for some period of time following the successful completion of a name identifier change. Not doing so could
1265 result in the inability of the Principal to access resources.

1266 If <RelayState> contains a value, the recipient **MUST** include this value in unmodified form in the
1267 <RelayState> element of the response.

1268 <AffiliationId>, if present, **MUST** be the Affiliation ID of a known affiliation with which the identity
1269 provider has an established relationship and of which the service provider is a member. If present, providers
1270 **MUST** refer to the federation associated with the specified affiliation, if any, not the service provider, in
1271 processing the message.

1272 **3.4 Federation Termination Notification Protocol**

1273 When the Principal terminates an identity federation between a service provider and an identity provider from the
1274 service provider, the service provider **MUST** send a <FederationTerminationNotification> message to
1275 the identity provider. The service provider is stating that it will no longer accept authentication assertions from the
1276 identity provider for the specified Principal.

1277 Likewise, when the Principal terminates an identity federation from the identity provider, the identity provider **MUST**
1278 send a <FederationTerminationNotification> message to the service provider. In this case, the identity
1279 provider is stating that it will no longer provide authentication assertions to the service provider for the specified
1280 Principal.

1281 This notification message is a one-way asynchronous message. Reasonable, best-effort delivery **MUST** be employed
1282 by all providers sending this message.

1283 **3.4.1 Message**

1284 The provider sends a <FederationTerminationNotification> to the provider with which it is terminating
1285 a federation.

1286 The <FederationTerminationNotification> message SHOULD be signed.

1287 **3.4.1.1 Element <FederationTerminationNotification>**

1288 The elements are as follows:

1289 Extension [Optional]

1290 Optional container for protocol extensions established by agreement between providers.

1291 ProviderID [Required]

1292 The identifier of the provider that is sending this message.

1293 NameIdentifier [Required]

1294 The name identifier of the Principal terminating federation. This name identifier MUST be equal to the
1295 <saml:NameIdentifier> element (and its included attributes) agreed upon earlier between the two
1296 communicating providers.

1297 id [Optional]

1298 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1299 AffiliationId [Optional]

1300 If present and the requester is a service provider, indicates that the requesting service provider is acting as a
1301 member of the specified Affiliation. If present and the requester is an identity provider, indicates that the receiving
1302 service provider should interpret the message as a member of the specified affiliation.

1303 consent [Optional]

1304 Indicates that consent has been obtained from a user in sending this message.

1305 The schema fragment is as follows:

```
1306 <element name="FederationTerminationNotification"  
1307 type="lib:FederationTerminationNotificationType"/>  
1308 <complexType name="FederationTerminationNotificationType">  
1309 <complexContent>  
1310 <extension base="samlp:RequestAbstractType">  
1311 <sequence>  
1312 <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>  
1313 <element ref="lib:ProviderID"/>  
1314 <element ref="saml:NameIdentifier"/>  
1315 <element ref="lib:AffiliationId" minOccurs="0"/>  
1316 </sequence>  
1317 <attribute name="id" type="ID" use="optional"/>  
1318 <attribute ref="lib:consent" use="optional"/>  
1319 </extension>  
1320 </complexContent>  
1321 </complexType>
```

1322 **3.4.1.2 Example**

```
1323 <FederationTerminationNotification id="12345" RequestID="9ec2-eb65-4bce-ab8f-  
1324 4becdf229815" MajorVersion="1" MinorVersion="2"  
1325 consent="urn:liberty:consent:obtained" IssueInstant="2001-12-  
1326 17T09:30:47Z">
```

```
1327     <ds:Signature>...</ds:Signature>  
1328     <ProviderID>http://IdentityProvider.com</ProviderID>  
1329     <saml:NameIdentifier>e958019a</saml:NameIdentifier>  
1330     <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>  
1331 </FederationTerminationNotification>
```

1332 3.4.2 Processing Rules

1333 The receiving provider **MUST** validate any signature present on the message. The signature on the message
1334 **MUST** be the signature of the <ProviderID> contained in the message. If the signature is not valid, the
1335 provider **MUST** ignore the message.

1336 If a provider receives a federation termination notification message that refers to a federation that does not exist
1337 from the perspective of the provider, the provider **MUST** ignore the message. Otherwise, the provider **MAY**
1338 perform any maintenance with the knowledge that the federation has been terminated.

1339 A provider **MAY** choose to invalidate the session of a user for whom federation has been terminated.

1340 <AffiliationID>, if present, **MUST** be the Affiliation ID of a known affiliation with which the identity
1341 provider has an established relationship and of which the service provider is a member. If present, providers
1342 **MUST** refer to the federation associated with the specified affiliation, if any, not the service provider, in
1343 processing the message.

1344 3.5 Single Logout Protocol

1345 The Single Logout Protocol provides a message exchange protocol by which all sessions authenticated by a particular
1346 identity provider are near-simultaneously terminated. The Single Logout Protocol is used either when a Principal logs
1347 out at a service provider or when the Principal logs out at an identity provider.

1348 When the Principal invokes the single logout process at a service provider, the service provider **MUST** send a
1349 <LogoutRequest> message to the identity provider that provided the authentication service for the session.

1350 When either the Principal invokes a logout at the identity provider or a service provider sends a logout request to the
1351 identity provider specifying that Principal, the identity provider **MUST** send a <LogoutRequest> message to each
1352 service provider to which it provided authentication assertions in the current session with the Principal, with the
1353 exception of the service provider that sent the <LogoutRequest> message to the Identity Provider.

1354 If the identity provider is *proxying* authentication from a second identity provider, then it **MUST** send a
1355 <LogoutRequest> to the *proxied* identity provider, unless the proxying provider has already received a
1356 <LogoutRequest> from the proxied provider.

1357 If the identity provider has provided authentication assertions on behalf of a Principal to a proxying identity provider,
1358 then it **MUST** send a <LogoutRequest> to that provider, unless the proxying provider has already received a
1359 <LogoutRequest> from the proxied provider.

1360 3.5.1 Request

1361 The <LogoutRequest> message indicates to the message receiver that a Principal's session was terminated. The
1362 message includes an optional <SessionIndex> element that **MUST** be specified if and only if the authentication
1363 statement in the assertion that the service provider used in establishing the session with the Principal contained a
1364 SessionIndex attribute. This message **SHOULD** be signed.

1365 3.5.1.1 Element <LogoutRequest>

1366 Extension [Optional]

1367 Optional container for protocol extensions established by agreement between providers. Implementors should note
1368 that this element may not contain content from the core Liberty namespace (which is prevented at the schema level
1369 by requiring namespace="##other").

1370 **NameIdentifier** [Required]

1371 The name identifier of the Principal that logged out. This name identifier **MUST** be equal to the
1372 <saml:NameIdentifier> element (including the equality of contained attributes) agreed upon between the
1373 two communicating providers.

1374 **ProviderID** [Required]

1375 The identifier of the provider that is making the request.

1376 **SessionIndex** [Optional]

1377 The session index specified in the authentication statement of the assertion used to establish the session being
1378 terminated. If a <SessionIndex> element was present in the authentication statement, an identical
1379 <SessionIndex> **MUST** be present in the <LogoutRequest>. If no <SessionIndex> element was
1380 present in the authentication statement, the <SessionIndex> **MUST** be omitted from the
1381 <LogoutRequest>.

1382 **id** [Optional]

1383 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1384 **RelayState** [Optional]

1385 This may contain state information that will be relayed back in the response. This data **SHOULD** be integrity-
1386 protected by the request author and **MAY** have other protections placed on it by the request author. An example of
1387 such protection is confidentiality.

1388 **AffiliationId** [Optional]

1389 If present and the requester is a service provider, indicates that the requesting service provider is acting as a
1390 member of the specified Affiliation. If present and the requester is an identity provider, indicates that the receiving
1391 service provider should interpret the message as a member of the specified affiliation.

1392 **consent** [Optional]

1393 Indicates that consent has been obtained from a user in sending this message.

1394 The schema fragment is as follows:

```
1395 <element name="LogoutRequest" type="lib:LogoutRequestType"/>
1396 <complexType name="LogoutRequestType">
1397   <complexContent>
1398     <extension base="samlp:RequestAbstractType">
1399       <sequence>
1400         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1401         <element ref="lib:ProviderID"/>
1402         <element ref="saml:NameIdentifier"/>
1403         <element name="SessionIndex" type="string" minOccurs="0"/>
1404         <element ref="lib:RelayState" minOccurs="0"/>
1405         <element ref="lib:AffiliationId" minOccurs="0"/>
1406       </sequence>
1407       <attribute name="id" type="ID" use="optional"/>
1408       <attribute ref="lib:consent" use="optional"/>
1409     </extension>
1410   </complexContent>
1411 </complexType>
```

1412 **3.5.1.2 Example**

```
1413 <LogoutRequest id="12345" RequestID="47693d03-7c33-4d65-931f-ddeb19fa6a73"  
1414     MajorVersion="1" MinorVersion="2" consent="urn:liberty:consent:obtained"  
1415     IssueInstant="2001-12-17T09:30:47Z">  
1416   <ds:Signature>...</ds:Signature>  
1417   <ProviderID>http://ServiceProvider.com</ProviderID>  
1418   <saml:NameIdentifier>342ad3d8-93ee-4c68-be35-cc9e7db39e2b</saml:NameIdentifier>  
1419   <SessionIndex>3</SessionIndex>  
1420   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>  
1421 </LogoutRequest>  
1422
```

1423 **3.5.1.3 Response**

1424 The responding provider MUST return a <LogoutResponse> message, which is of type **StatusResponseType**.

1425 This message SHOULD be signed.

1426 **3.5.1.4 Element <LogoutResponse>**

1427 The elements of the message are as follows:

1428 **Extension** [Optional]

1429 Optional container for protocol extensions established by agreement between providers.

1430 **ProviderID** [Required]

1431 The identifier of the provider responding.

1432 **Status** [Required]

1433 A status code that indicates the result of the request.

1434 **id** [Optional]

1435 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1436 **RelayState** [Optional]

1437 This contains state information that may have appeared in the request, and is being relayed back to the sender.

1438 The schema fragment is as follows:

```
1439 <element name="LogoutResponse" type="lib:StatusResponseType"/>
```

1440 **3.5.1.5 Example**

```
1441 <LogoutResponse id="12345" ResponseID="74ffec0f-1165-4fa3-b088-3dd2c2388b91"  
1442     InResponseTo="eb20e77f-d982-44f9-936e-dd135bf437d4" MajorVersion="1"  
1443     MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z"  
1444     Recipient="http://ServiceProvider.com">  
1445   <ds:Signature>...</ds:Signature>  
1446   <ProviderID>http://IdentityProvider.com</ProviderID>  
1447   <samlp:Status>  
1448     <samlp:StatusCode Value="samlp:Success" />  
1449   </samlp:Status>  
1450   <RelayState>R0lGODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>  
1451 </LogoutResponse>
```

1452 **3.5.1.6 Processing Rules**

1453 If <RelayState> contains a value, the recipient MUST include this value in unmodified form in the
1454 <RelayState> element of the response.

1455 <AffiliationId>, if present, MUST be the Affiliation ID of a known affiliation with which the identity
1456 provider has an established relationship and of which the service provider is a member. If present, providers
1457 MUST interpret the <saml:NameIdentifier> in the context of the specified affiliation, not the service
1458 provider, in processing the message.

1459 Other unique processing rules apply based on whether the message receiver is an identity provider or a service
1460 provider.

1461 **3.5.1.7 Identity Provider Processing Rules**

1462 When an identity provider receives the <LogoutRequest> message, the identity provider MUST validate that any
1463 signature present on the message is the signature of a service provider to which the identity provider provided an
1464 authentication assertion for the current session. If that holds, the identity provider SHOULD do the following:

1465 Send a <LogoutRequest> message to each service provider for which the identity provider provided
1466 authentication assertions in the current session, *other than the originator of the <LogoutRequest>*.

1467 Send a <LogoutRequest> message to the identity provider on behalf of whom the identity provider
1468 proxied the user's authentication, *unless the second identity provider is the originator of the*
1469 *<LogoutRequest>*.

1470 Terminate the Principal's current session as specified by the <saml:NameIdentifier> element.

1471 If an error occurs during this further processing of the logout (for example, relying service providers may not
1472 all implement the Single Logout profile used by the requesting service provider), then the identity
1473 provider MUST respond to the original requestor with a <LogoutResponse> message, indicating the
1474 status of the logout request. The value "lib:UnsupportedProfile" is provided for a second-level
1475 <samlp:StatusCode>, indicating that a service provider should retry the <LogoutRequest>
1476 using a different profile.

1477 **3.5.1.8 Service Provider Processing Rules**

1478 When the service provider receives the <LogoutRequest> message, the service provider MUST validate the
1479 identity provider's signature contained in the <ds:Signature> element. If the signature is that of the identity
1480 provider that provided the authentication for the Principal's current session, the service provider MUST invalidate the
1481 Principal's session referred to in the <saml:NameIdentifier> element.

1482 **3.6 Introduction Notification Protocol**

1483 If a service provider is successfully introduced to an identity provider through the mediation of an *introducing* identity
1484 provider; the service provider federates a Principal's identity with the provider to whom they have been introduced,
1485 and if the introducing provider has requested notification of federation by including a <Notification> element of
1486 "true" in the <IntroductionStatement>, then the identity provider to whom the service provider has been
1487 introduced MUST send an <IntroductionNotification> message to the introducing provider.

1488 This notification message is a one-way asynchronous message. Reasonable, best-effort delivery MUST be employed
1489 by all providers sending this message.

1490 **3.6.1 Message**

1491 An identity provider that is federating a Principal's identity as a result of the introduction of another identity provider
1492 sends a <IntroductionNotification> message to the introducing provider if that provider has requested
1493 introduction notification via the <Notification> element.

1494 The <IntroductionNotification> message SHOULD be signed.

1495 **3.6.1.1 Element <IntroductionNotification>**

1496 The elements are as follows:

1497 Extension [Optional]

1498 Optional container for protocol extensions established by agreement between providers.

1499 ProviderID [Required]

1500 The identifier of the identity provider that is sending this message.

1501 IntroducedProviderID or IntroducedAffiliationID [Required]

1502 The identifier of the service provider or affiliation group that has been successfully introduced.

1503 AssertionIDReference [Required]

1504 The identifier of the introduction assertion utilized by the sending provider to accept the introduction of the
1505 provider to whom the notification is being sent.

1506 NameIdentifier [Required]

1507 The name identifier of the Principal whose identity federation triggered the introduction. This name identifier
1508 MUST be unique per Principal, generated specifically for use by this protocol and MUST be different for each
1509 introducing identity provider. It MUST be reused if the same Principal triggers introduction of subsequent service
1510 providers.

1511 id [Optional]

1512 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1513 The schema fragment is as follows:

```
1514 <element name="IntroductionNotification" type="lib:IntroductionNotificationType"/>
1515 <complexType name="IntroductionNotificationType">
1516   <complexContent>
1517     <extension base="samlp:RequestAbstractType">
1518       <sequence>
1519         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1520         <element ref="lib:ProviderID"/>
1521         <choice>
1522           <element name="IntroducedProviderID" type="lib:ProviderIDType"/>
1523           <element name="IntroducedAffiliationID" type="lib:ProviderIDType"/>
1524         </choice>
1525         <element ref="saml:AssertionIDReference"/>
1526         <element ref="saml:NameIdentifier"/>
1527       </sequence>
1528       <attribute name="id" type="ID" use="optional"/>
1529     </extension>
1530   </complexContent>
1531 </complexType>
```

1532 **3.6.1.2 Example**

```
1533 <IntroductionNotification id="12345" RequestID="9ec2-eb65-4bce-ab8f-4becdf229815"  
1534     MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-17T09:30:47Z">  
1535     <ds:Signature>...</ds:Signature>  
1536     <ProviderID>http://IdentityProvider.com</ProviderID>  
1537     <IntroducedProviderID>http://ServiceProvider.com</IntroducedProviderID>  
1538     <saml:AssertionIDReference>dswd398t4deurd4</saml:AssertionIDReference>  
1539     <saml:NameIdentifier>e958019a</saml:NameIdentifier>  
1540 </IntroductionNotification>
```

1541 **3.6.2 Processing Rules**

1542 The receiving identity provider **MUST** validate any signature present on the message. The signature on the message
1543 **MUST** be the signature of the <ProviderID> contained in the message. If the signature is not valid, the provider
1544 **MUST** ignore the message.

1545 **3.7 Provider Relationship Termination Protocol**

1546 When a service provider has previously been successfully introduced to an identity provider through the mediation of
1547 an introducing identity provider, but the business relationship between the identity providers has subsequently been
1548 broken, then the introducing identity provider **SHOULD** send a
1549 <ProviderRelationshipTerminationRequest> message to the service provider.

1550 Upon reception of a <ProviderRelationshipTerminationRequest> message, a provider **MUST** respond
1551 with a <ProviderRelationshipTerminationResponse> message.

1552 **3.7.1 Message**

1553 The introducing identity provider sends a <ProviderRelationshipTerminationRequest> to the service
1554 provider concerning the termination of the relationship between the identity providers.

1555 The <ProviderRelationshipTerminationRequest> message **MUST** be signed.

1556 **3.7.1.1 Element <ProviderRelationshipTerminationRequest>**

1557 The elements are as follows:

1558 Extension [Optional]

1559 Optional container for protocol extensions established by agreement between providers.

1560 ProviderID [Required]

1561 The identifier of the provider that is sending this message.

1562 TerminatedProviderID [Required]

1563 The identifier of the provider with which the sender has terminated a relationship.

1564 id [Optional]

1565 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1566 RelayState [Optional]

1567 This contains state information that will be relayed back in the response. This data **SHOULD** be integrity-protected
1568 by the request author and **MAY** have other protections placed on it by the request author. An example of such
1569 protection is confidentiality.

1570 The schema fragment is as follows:

```

1571 <element name="ProviderRelationshipTerminationRequest"
1572 type="lib:ProviderRelationshipTerminationRequestType"/>
1573 <complexType name="ProviderRelationshipTerminationRequestType">
1574 <complexContent>
1575 <extension base="sampl:RequestAbstractType">
1576 <sequence>
1577 <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1578 <element ref="lib:ProviderID"/>
1579 <element name="TerminatedProviderID" type="lib:ProviderIDType"/>
1580 <element ref="lib:RelayState" minOccurs="0"/>
1581 </sequence>
1582 <attribute name="id" type="ID" use="optional"/>
1583 </extension>
1584 </complexContent>
1585 </complexType>

```

1586 3.7.1.2 Example

```

1587 <ProviderRelationshipTerminationRequest id="abcde" RequestID="9ec2-eb65-4bce-ab8f-
1588 4becdf229815" MajorVersion="1" MinorVersion="2" IssueInstant="2001-12-
1589 17T09:30:47Z">
1590 <ds:Signature>...</ds:Signature>
1591 <ProviderID>http://IntroducingIdentityProvider.com</ProviderID>
1592 <TerminatedProviderID>http://IntroducedIdentityProvider.com</TerminatedProvider
1593 ID>
1594 <RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1595 </ProviderRelationshipTerminationRequest>

```

1596 3.7.2 Response

1597 The responding provider MUST return a <ProviderRelationshipTerminationResponse> message,
1598 which is of type **StatusResponseType**.

1599 This message SHOULD be signed.

1600 3.7.2.1 Element <ProviderRelationshipTerminationResponse>

1601 The elements of the message are as follows:

1602 **Extension** [Optional]

1603 Optional container for protocol extensions established by agreement between providers.

1604 **ProviderID** [Required]

1605 The identifier of the provider responding.

1606 **Status** [Required]

1607 A status code that indicates the reception of the notification message.

1608 **id** [Optional]

1609 Identifier used to identify this element in the signature. See section 3.1.5, Signature Verification.

1610 **RelayState** [Optional]

1611 This contains state information that may have appeared in the request, and is being relayed back to the sender.

1612 The schema fragment is as follows:

```

1613 <element name="ProviderRelationshipTerminationResponse"
1614 type="lib>StatusResponseType"/>

```

1615 **3.7.2.2 Example**

```
1616 <ProviderRelationshipTerminationResponse id="abcde" ResponseID="74ffec0f-1165-
1617 4fa3-b088-3dd2c2388b91" InResponseTo="eb20e77f-d982-44f9-936e-
1618 dd135bf437d4" MajorVersion="1" MinorVersion="0" IssueInstant="2001-12-
1619 17T09:30:47Z" Recipient="http://IntroducingIdentityProvider.com">
1620 <ds:Signature>...</ds:Signature>
1621 <ProviderID>http://ServiceProvider.com</ProviderID>
1622 <samlp:Status>
1623 <samlp:StatusCode Value="samlp:Success"/>
1624 </samlp:Status>
1625 <RelayState>R01GODlhcgGSALMAAAQCAEMmCZtuMFQxDS8b</RelayState>
1626 </ProviderRelationshipTerminationResponse>
```

1627 **3.7.3 Processing Rules**

1628 The receiving provider MUST validate any signature present on the message. The signature on the message
1629 MUST be the signature of the <ProviderID> contained in the message. If the signature is not valid, the
1630 provider MUST ignore the message.

1631 If <RelayState> contains a value, the recipient MUST include this value in unmodified form in the
1632 <RelayState> element of the response.

1633 **3.7.3.1 Recipient Processing Rules**

1634 The receiving provider MAY choose to invalidate the relationship with the provider identified by the
1635 <TerminatedProviderID> element.

1636 The recipient MUST respond with a <ProviderRelationshipTerminationResponse> message.

1637 **4 Schema Definition**

```
1638 <?xml version="1.0" encoding="UTF-8"?>
1639 <schema targetNamespace="urn:liberty:iff:1.2" xmlns:lib="urn:liberty:iff:1.2"
1640 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1641 xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
1642 xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:ac="urn:liberty:ac:1.2"
1643 xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
1644 attributeFormDefault="unqualified">
1645 <import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
1646 schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-
1647 schema-assertion-01.xsd"/>
1648 <import namespace="urn:oasis:names:tc:SAML:1.0:protocol"
1649 schemaLocation="http://www.oasis-open.org/committees/security/docs/cs-sstc-
1650 schema-protocol-01.xsd"/>
1651 <import namespace="http://www.w3.org/2000/09/xmldsig#"
1652 schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
1653 <import namespace="urn:liberty:ac:1.2"
1654 schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-
1655 authentication-context-v1.2.xsd"/>
1656 <include schemaLocation="http://www.projectliberty.org/specs/liberty-architecture-
1657 utility-v1.0.xsd"/>
1658 <simpleType name="ProviderIDType">
1659 <restriction base="anyURI"/>
1660 </simpleType>
1661 <element name="ProviderID" type="lib:ProviderIDType"/>
1662 <element name="AffiliationID" type="lib:ProviderIDType"/>
1663 <complexType name="SignedSAMLRequestType">
1664 <complexContent>
1665 <extension base="samlp:RequestType">
1666 <attribute name="id" type="ID" use="optional"/>
1667 </extension>
```

```
1668     </complexContent>
1669 </complexType>
1670 <element name="AuthnRequest" type="lib:AuthnRequestType"/>
1671 <complexType name="AuthnRequestType">
1672   <complexContent>
1673     <extension base="samlp:RequestAbstractType">
1674       <sequence>
1675         <element ref="lib:Extension minOccurs="0" maxOccurs="unbounded"/>
1676         <element ref="lib:ProviderID"/>
1677         <element ref="lib:AffiliationId minOccurs="0"/>
1678         <element ref="lib:NameIDPolicy minOccurs="0"/>
1679         <element name="ForceAuthn" type="boolean" minOccurs="0"/>
1680         <element name="IsPassive" type="boolean" minOccurs="0"/>
1681         <element ref="lib:ProtocolProfile minOccurs="0"/>
1682         <element name="AssertionConsumerServiceID" type="string" minOccurs="0"/>
1683         <element ref="lib:AuthnContext minOccurs="0"/>
1684         <element ref="lib:RelayState minOccurs="0"/>
1685         <element name="AuthnContextComparison"
1686           type="lib:AuthnContextComparisonType minOccurs="0"/>
1687         <element name="ProxyCount" type="nonNegativeInteger" minOccurs="0"/>
1688         <element ref="lib:IntroductionArtifact minOccurs="0"/>
1689       </sequence>
1690       <attribute name="id" type="ID" use="optional"/>
1691       <attribute ref="lib:consent" use="optional"/>
1692     </extension>
1693   </complexContent>
1694 </complexType>
1695 <simpleType name="NameIDPolicyType">
1696   <restriction base="string">
1697     <enumeration value="none"/>
1698     <enumeration value="temporary"/>
1699     <enumeration value="federated"/>
1700     <enumeration value="any"/>
1701   </restriction>
1702 </simpleType>
1703 <simpleType name="AuthnContextComparisonType">
1704   <restriction base="string">
1705     <enumeration value="exact"/>
1706     <enumeration value="minimum"/>
1707     <enumeration value="better"/>
1708   </restriction>
1709 </simpleType>
1710 <element name="NameIDPolicy" type="lib:NameIDPolicyType"/>
1711 <element name="RelayState" type="string"/>
1712 <element name="ProtocolProfile" type="anyURI"/>
1713 <element name="AuthnContext">
1714   <complexType>
1715     <choice>
1716       <element name="AuthnContextClassRef" type="anyURI" maxOccurs="unbounded"/>
1717       <element name="AuthnContextStatementRef" type="anyURI"
1718         maxOccurs="unbounded"/>
1719     </choice>
1720   </complexType>
1721 </element>
1722 <element name="IntroductionArtifact" type="string"/>
1723 <element name="AuthnResponse" type="lib:AuthnResponseType"/>
1724 <complexType name="AuthnResponseType">
1725   <complexContent>
1726     <extension base="samlp:ResponseType">
1727       <sequence>
1728         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1729         <element ref="lib:ProviderID"/>
1730         <element ref="lib:RelayState minOccurs="0"/>
1731       </sequence>
1732       <attribute name="id" type="ID" use="optional"/>
1733       <attribute ref="lib:consent" use="optional"/>
1734     </extension>
```

```
1735     </complexContent>
1736 </complexType>
1737 <element name="Assertion" type="lib:AssertionType"/>
1738 <complexType name="AssertionType">
1739   <complexContent>
1740     <extension base="saml:AssertionType">
1741       <attribute name="InResponseTo" type="saml:IDReferenceType"/>
1742       <attribute name="id" type="ID" use="optional"/>
1743     </extension>
1744   </complexContent>
1745 </complexType>
1746 <complexType name="SubjectType">
1747   <complexContent>
1748     <extension base="saml:SubjectType">
1749       <sequence>
1750         <element ref="lib:IDPProvidedNameIdentifier"/>
1751       </sequence>
1752     </extension>
1753   </complexContent>
1754 </complexType>
1755 <complexType name="AuthenticationStatementType">
1756   <complexContent>
1757     <extension base="saml:AuthenticationStatementType">
1758       <sequence>
1759         <element name="AuthnContext" minOccurs="0">
1760           <complexType>
1761             <sequence>
1762               <element name="AuthnContextClassRef" type="anyURI" minOccurs="0"/>
1763               <choice>
1764                 <element ref="ac:AuthenticationContextStatement"/>
1765                 <element name="AuthnContextStatementRef" type="anyURI"/>
1766               </choice>
1767             </sequence>
1768           </complexType>
1769         </element>
1770       </sequence>
1771       <attribute name="ReauthenticateOnOrAfter" type="dateTime" use="optional"/>
1772       <attribute name="SessionIndex" type="string" use="optional"/>
1773     </extension>
1774   </complexContent>
1775 </complexType>
1776 <complexType name="IntroductionStatementType">
1777   <complexContent>
1778     <extension base="saml:SubjectStatementType">
1779       <choice maxOccurs="unbounded">
1780         <element ref="lib:ProviderID"/>
1781         <element ref="lib:AffiliationID"/>
1782       </choice>
1783       <element name="Notification" type="boolean" minOccurs="0"/>
1784     </extension>
1785   </complexContent>
1786 </complexType>
1787 <element name="AuthnRequestEnvelope" type="lib:AuthnRequestEnvelopeType"/>
1788 <complexType name="AuthnRequestEnvelopeType">
1789   <complexContent>
1790     <extension base="lib:RequestEnvelopeType">
1791       <sequence>
1792         <element ref="lib:AuthnRequest"/>
1793         <element ref="lib:ProviderID"/>
1794         <element name="ProviderName" type="string" minOccurs="0"/>
1795         <element name="AssertionConsumerServiceURL" type="anyURI"/>
1796         <element ref="lib:IDPList" minOccurs="0"/>
1797         <element name="IsPassive" type="boolean" minOccurs="0"/>
1798       </sequence>
1799     </extension>
1800   </complexContent>
1801 </complexType>
```

```
1802 <complexType name="RequestEnvelopeType">
1803   <sequence>
1804     <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1805   </sequence>
1806 </complexType>
1807 <element name="IDPLList" type="lib:IDPLListType"/>
1808 <complexType name="IDPLListType">
1809   <sequence>
1810     <element ref="lib:IDPEntries"/>
1811     <element ref="lib:GetComplete" minOccurs="0" maxOccurs="unbounded"/>
1812   </sequence>
1813 </complexType>
1814 <element name="IDPEntries">
1815   <complexType>
1816     <sequence>
1817       <element ref="lib:ProviderID"/>
1818       <element name="ProviderName" type="string" minOccurs="0"/>
1819       <element name="Loc" type="anyURI"/>
1820     </sequence>
1821   </complexType>
1822 </element>
1823 <element name="IDPEntries">
1824   <complexType>
1825     <sequence>
1826       <element ref="lib:IDPEntries" maxOccurs="unbounded"/>
1827     </sequence>
1828   </complexType>
1829 </element>
1830 <element name="GetComplete" type="anyURI"/>
1831 <element name="AuthnResponseEnvelope" type="lib:AuthnResponseEnvelopeType"/>
1832 <complexType name="AuthnResponseEnvelopeType">
1833   <complexContent>
1834     <extension base="lib:ResponseEnvelopeType">
1835       <sequence>
1836         <element ref="lib:AuthnResponse"/>
1837         <element name="AssertionConsumerServiceURL" type="anyURI"/>
1838       </sequence>
1839     </extension>
1840   </complexContent>
1841 </complexType>
1842 <complexType name="ResponseEnvelopeType">
1843   <sequence>
1844     <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1845   </sequence>
1846 </complexType>
1847 <element name="RegisterNameIdentifierRequest"
1848   type="lib:RegisterNameIdentifierRequestType"/>
1849 <complexType name="RegisterNameIdentifierRequestType">
1850   <complexContent>
1851     <extension base="samlp:RequestAbstractType">
1852       <sequence>
1853         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded"/>
1854         <element ref="lib:ProviderID"/>
1855         <element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1856         <element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1857         <element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1858         <element ref="lib:RelayState" minOccurs="0"/>
1859         <element ref="lib:AffiliationId" minOccurs="0"/>
1860       </sequence>
1861       <attribute name="id" type="ID" use="optional"/>
1862     </extension>
1863   </complexContent>
1864 </complexType>
1865 <element name="IDPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1866 <element name="SPProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1867 <element name="OldProvidedNameIdentifier" type="saml:NameIdentifierType"/>
1868 <element name="RegisterNameIdentifierResponse" type="lib:StatusResponseType"/>
```

```

1869 <complexType name="StatusResponseType">
1870 <complexContent>
1871 <extension base="samlp:ResponseAbstractType">
1872 <sequence>
1873 <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1874 <element ref="lib:ProviderID" />
1875 <element ref="samlp:Status" />
1876 <element ref="lib:RelayState" minOccurs="0" />
1877 </sequence>
1878 <attribute name="id" type="ID" use="optional" />
1879 </extension>
1880 </complexContent>
1881 </complexType>
1882 <element name="FederationTerminationNotification"
1883 type="lib:FederationTerminationNotificationType" />
1884 <complexType name="FederationTerminationNotificationType">
1885 <complexContent>
1886 <extension base="samlp:RequestAbstractType">
1887 <sequence>
1888 <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1889 <element ref="lib:ProviderID" />
1890 <element ref="saml:NameIdentifier" />
1891 <element ref="lib:AffiliationId" minOccurs="0" />
1892 </sequence>
1893 <attribute name="id" type="ID" use="optional" />
1894 <attribute ref="lib:consent" use="optional" />
1895 </extension>
1896 </complexContent>
1897 </complexType>
1898 <element name="LogoutRequest" type="lib:LogoutRequestType" />
1899 <complexType name="LogoutRequestType">
1900 <complexContent>
1901 <extension base="samlp:RequestAbstractType">
1902 <sequence>
1903 <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1904 <element ref="lib:ProviderID" />
1905 <element ref="saml:NameIdentifier" />
1906 <element name="SessionIndex" type="string" minOccurs="0" />
1907 <element ref="lib:RelayState" minOccurs="0" />
1908 <element ref="lib:AffiliationId" minOccurs="0" />
1909 </sequence>
1910 <attribute name="id" type="ID" use="optional" />
1911 <attribute ref="lib:consent" use="optional" />
1912 </extension>
1913 </complexContent>
1914 </complexType>
1915 <element name="LogoutResponse" type="lib:StatusResponseType" />
1916 <element name="IntroductionNotification" type="lib:IntroductionNotificationType" />
1917 <complexType name="IntroductionNotificationType">
1918 <complexContent>
1919 <extension base="samlp:RequestAbstractType">
1920 <sequence>
1921 <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1922 <element ref="lib:ProviderID" />
1923 <choice>
1924 <element name="IntroducedProviderID" type="lib:ProviderIDType" />
1925 <element name="IntroducedAffiliationID" type="lib:ProviderIDType" />
1926 </choice>
1927 <element ref="saml:AssertionIDReference" />
1928 <element ref="saml:NameIdentifier" />
1929 </sequence>
1930 <attribute name="id" type="ID" use="optional" />
1931 </extension>
1932 </complexContent>
1933 </complexType>
1934 <element name="ProviderRelationshipTerminationRequest"
1935 type="lib:ProviderRelationshipTerminationRequestType" />

```

```
1936 <complexType name="ProviderRelationshipTerminationRequestType">
1937   <complexContent>
1938     <extension base="sampl:RequestAbstractType">
1939       <sequence>
1940         <element ref="lib:Extension" minOccurs="0" maxOccurs="unbounded" />
1941         <element ref="lib:ProviderID" />
1942         <element name="TerminatedProviderID" type="lib:ProviderIDType" />
1943         <element ref="lib:RelayState" minOccurs="0" />
1944       </sequence>
1945       <attribute name="id" type="ID" use="optional" />
1946     </extension>
1947   </complexContent>
1948 </complexType>
1949 <element name="ProviderRelationshipTerminationResponse"
1950   type="lib:StatusResponseType" />
1951 </schema>
1952
```

1952 **5 References**

- 1953 [LibertyBindProf] Rouault, J., & Wason, T., eds. (January 2003). “Liberty Bindings and Profiles
1954 Specification,” Version 1.2. Liberty Alliance Project,
1955 <<http://www.projectliberty.org/specs/>>.
- 1956 [LibertyGloss] Mauldin, H., & Wason, T., eds. (January 2003). “Liberty Architecture Glossary,” Version
1957 1.1. Liberty Alliance Project, <<http://www.projectliberty.org/specs/>>.
- 1958 [RFC2119] Bradner, S. (March 1997). “Key words for use in RFCs to Indicate Requirement Levels,”
1959 RFC 2119. The Internet Engineering Task Force, <[http://www.rfc-
1960 editor.org/rfc/rfc2119.txt](http://www.rfc-editor.org/rfc/rfc2119.txt)> [18 December 2002].
- 1961 [RFC3280] Housley, R. (April 2002). “Internet X.509 Public Key Infrastructure Certificate and
1962 Certificate Revocation List (CRL) Profile,” RFC 3280. The Internet Engineering Task
1963 Force, <<http://www.rfc-editor.org/rfc/rfc3280.txt>> [18 December 2002]
- 1964 [SAMLCore] Hallam-Baker, P., Maler, E., eds. (05 Nov. 2002). “Assertions and Protocol for the OASIS
1965 Security Assertion Markup Language (SAML),” Version 1.0, OASIS Standard.
1966 Organization for the Advancement of Structured Information Standards, <[http://www.oasis-
1967 open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)> [18 December 2002].
- 1968 [SAMLBind] Mishra, P., ed.. (05 Nov. 2002). “Bindings and Profiles for the OASIS Security Assertion
1969 Markup Language (SAML),” Version 1.0, OASIS Standard. Organization for the
1970 Advancement of Structured Information Standards, <[http://www.oasis-
1971 open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)> [18 December 2002].
- 1972 [Schema1] Thompson, H. S., Beech, D., Maloney, M., & Mendleson, N., eds. (May 2002). “XML
1973 Schema Part 1: Structures,” Recommendation. World Wide Web Consortium,
1974 <<http://www.w3.org/TR/xmlschema-1/>> [18 December 2002].
- 1975 [Schema2] Biron, P. V., & Malhotra, A., eds. (May 2002). “XML Schema Part 2: Datatypes,”
1976 Recommendation. World Wide Web Consortium, <<http://www.w3.org/TR/xmlschema-2/>>
1977 [18 December 2002].
- 1978 [XMLDsig] D. Eastlake, J. Reagle, D. Solo et al, (12 February 2002). “XML-Signature Syntax and
1979 Processing”, Recommendation. World Wide Web Consortium,
1980 <<http://www.w3.org/TR/xmlsig-core>> [18 December 2002].
- 1981 [XMLCanon] J. Boyer, D. Eastlake, & J. Reagle (18 July 2002). “Exclusive XML Canonicalization,”
1982 Recommendation. World Wide Web Consortium, <<http://www.w3.org/TR/xml-exc-c14n>>
1983 [18 December 2002].
- 1984 [XPointer] DeRose, S., Maler, E., & Daniel, R. (16 August 2002). “XML Pointer Language
1985 (XPointer),” Working Draft. World Wide Web Consortium, <<http://www.w3.org/TR/xptr/>>
1986 [18 December 2002].
- 1987 [LibMetadata] Davis, P. (14 April 2003). “Liberty Core Metadata”, Version 1.0-DRAFT. Liberty Alliance
1988 Project, <<http://www.projectliberty.org/specs>> [14 April 2003].
- 1989