



Liberty ID-FF Bindings and Profiles Specification

Version: 1.2

Editors:

Scott Cantor, OSU/Internet2
John Kemp, IEEE-ISTO

Contributors:

Robert Aarts, Nokia
Slava Kavsan, RSA Security
Tom Wason, IEEE-ISTO

Abstract:

Specification of the Liberty Alliance Project core profiles and bindings. This specification defines the bindings and profiles of the Liberty protocols and messages to HTTP-based communication frameworks. This specification relies on the SAML core framework in SAML Core V1.1 and makes use of adaptations of the SAML profiles in SAML Bindings V1.1.

Filename: liberty-idff-bindings-profiles-v1.2.pdf

1 Notice

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

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

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

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

30 **Contents**

31 [1. Introduction](#) 4
32 [2. Protocol Bindings](#) 5
33 [3. Profiles](#) 11
34 [References](#) 60

35 1. Introduction

36 This specification defines the bindings and profiles of the Liberty protocols and messages to HTTP-based commu-
37 nication frameworks. This specification relies on the SAML core framework in [SAMLCore11] and makes use of
38 adaptations of the SAML profiles in [SAMLBind11]. A separate specification, [LibertyProtSchema], is used to define
39 the Liberty protocols and messages used within the profiles. Definitions for Liberty-specific terms can be found in
40 [LibertyGlossary].

41 1.1. Notation

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

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

49 Listings of productions or other normative code appear like this.

50 Example code listings appear like this.

51 **Note:**

52 Non-normative notes and explanations appear like this.

53 Conventional XML namespace prefixes are used throughout this specification to stand for their respective namespaces
54 as follows, regardless of whether a namespace declaration is present in the example:

55 XML Namespace Conventions

- 56 • The prefix `lib:` stands for the Liberty namespace `urn:liberty:iff:2003-08`
- 57 • The prefix `saml:` stands for the SAML assertion namespace (see [SAMLCore]).
- 58 • The prefix `samlp:` stands for the SAML request-response protocol namespace (see [SAMLCore]).
- 59 • The prefix `ds:` stands for the W3C XML signature namespace, `http://www.w3.org/2000/09/xmldsig#`
- 60 • The prefix `xenc:` stands for the W3C XML encryption namespace, `http://www.w3.org/2001/04/xmlenc#`
- 61 • The prefix `SOAP-ENV:` stands for the SOAP 1.1 namespace, `http://schemas.xmlsoap.org/soap/envelope`
62 (see [SOAP1.1]).

63 Terminology from [RFC2396] is used to describe components of an HTTP URL. An HTTP URL has the following
64 form:

65 `<scheme>://<authority><path>?<query>`

66 Sections in this document specify certain portions of the `<query>` component of the URL. Ellipses (...) are used to
67 indicate additional, but unspecified, portions of the `<query>` component.

68 **2. Protocol Bindings**

69 The Liberty protocol bindings are defined in this section.

70 **2.1. SOAP Binding for Liberty**

71 The Liberty SOAP binding defines how to use SOAP to send and receive Liberty protocol requests and responses using
72 SOAP 1.1 messages.

73 Like Liberty, SOAP can be used over multiple underlying transports. This binding has protocol- independent aspects,
74 but **REQUIRES** the use of SOAP over HTTP.

75 **2.1.1. Protocol-Independent Aspects of the Liberty SOAP Binding**

76 The following sections define aspects of the Liberty SOAP binding that are independent of the underlying protocol,
77 such as HTTP, on which the SOAP messages are transported.

78 **2.1.1.1. Basic Operation**

79 SOAP messages consist of three elements: an envelope, header data, and a message body. Liberty request-response
80 protocol elements **MUST** be enclosed within the SOAP message body.

81 SOAP 1.1 also defines an optional data encoding system. This system is not used within the Liberty SOAP binding.
82 This means that SAML messages can be transported using SOAP without re-encoding from the "standard" Liberty
83 schemas to one based on the SOAP encoding.

84 The specific profile determines the type of messages that can be sent or received. The system model used for Liberty
85 conversations over SOAP may be a simple request-response model, or it may be a more complex interaction that
86 includes HTML forms or other input mechanisms that interact with a Principal.

87 This Liberty specification defines constraints. Liberty protocol messages **MUST** be sent as the top level element in
88 the SOAP body. The requester or responder **MUST** not include more than one Liberty protocol message in a single
89 SOAP message. The requester or responder **MUST** not include any additional XML elements in the SOAP body.
90 Additionally, if a SOAP fault code is returned, then no Liberty protocol message may appear in the SOAP body. SOAP
91 faults **MUST** only be used for signaling non-Liberty-related errors.

92 [\[SOAPv1.1\]](#) references an early draft of the XML Schema specification including an obsolete namespace. Originators
93 of Liberty SOAP messages **SHOULD** generate SOAP messages referencing only the final XML schema namespace.
94 Receivers of Liberty SOAP messages **MUST** be able to process both the XML schema namespace used in [\[SOAPv1.1\]](#)
95 and the final XML schema namespace.

96 **2.1.1.2. SOAP Headers**

97 A Liberty SOAP message **MAY** contain arbitrary headers added to the SOAP message. This binding does not define
98 any additional SOAP headers.

99 Liberty SOAP messages **MUST NOT** require that any headers be understood for correct interpretation of the message.

100 **2.1.1.3. Authentication**

101 Authentication of Liberty messages is **OPTIONAL** and depends on the environment of use. Authentication protocols
102 available from the underlying substrate protocol **MAY** be utilized to provide authentication. Section [Section 2.1.2.1](#)
103 describes authentication in the SOAP-over-HTTP environment.

104 **2.1.1.4. Message Integrity**

105 Message integrity of Liberty messages is OPTIONAL and depends on the environment of use. The security layer in
106 the underlying substrate protocol MAY be used to ensure message integrity. Section [Section 2.1.2.2](#) describes support
107 for message integrity in the SOAP-over-HTTP environment.

108 **2.1.1.5. Confidentiality**

109 Confidentiality of Liberty messages is OPTIONAL and depends on the environment of use. The security layer in
110 the underlying substrate protocol MAY be used to ensure message confidentiality. Section [Section 2.1.2.3](#) describes
111 support for confidentiality in the SOAP over HTTP environment.

112 **2.1.2. Use of SOAP over HTTP**

113 This section describes certain specifics of using SOAP over HTTP, including HTTP headers, error reporting,
114 authentication, message integrity and confidentiality.

115 The HTTP binding for SOAP is described in [\[SOAPv1.1\]](#) §6.0. It requires the use of a SOAPAction header as part of
116 a SOAP HTTP request. Processing of a Liberty message MUST NOT depend on the value of this header. A Liberty
117 message MAY set the value of its SOAPAction header as follows:

```
118  
119     urn:liberty:soap-action  
120  
121
```

122 **2.1.2.1. Authentication**

123 Liberty SOAP message endpoints MUST implement the following authentication methods:

- 124 1. No client or server authentication.
- 125 2. HTTP basic client authentication [\[RFC2617\]](#) with and without SSL 3.0 or TLS 1.0.
- 126 3. HTTP over SSL 3.0 or TLS 1.0 (see Section 6) server authentication with a server-side certificate.
- 127 4. HTTP over SSL 3.0 or TLS 1.0 client authentication with a client-side certificate.

128 If a message receiver uses SSL 3.0 or TLS 1.0, it MUST use a server-side certificate.

129 **2.1.2.2. Message Integrity**

130 When message integrity needs to be guaranteed, messages MUST be sent with HTTP over SSL 3.0 or TLS1.0 with a
131 server-side certificate.

132 **2.1.2.3. Message Confidentiality**

133 When message confidentiality is required, messages MUST be sent with HTTP over SSL 3.0 or TLS 1.0 with a
134 server-side certificate.

135 **2.1.2.4. Security Considerations**

136 Before deployment in a given profile, each combination of authentication, message integrity and confidentiality
137 mechanisms SHOULD be analyzed for vulnerability in the context of the profile.

138 [\[RFC2617\]](#) describes possible attacks in the HTTP environment when basic or message- digest authentication schemes
139 are used.

140 **2.1.2.5. Error Reporting**

141 A message receiver that refuses to perform a message exchange SHOULD return a "403 Forbidden" response. In this
 142 case, the content of the HTTP body is not significant.

143 As described in [SOAPv1.1] § 6.2, in the case of a SOAP error while processing a SOAP request, the SOAP HTTP
 144 server MUST return a "500 Internal Server Error" response and include a SOAP message in the response with a SOAP
 145 fault element. This type of error SHOULD be returned for SOAP-related errors detected before control is passed to
 146 the Liberty message processor, or when the SOAP processor reports an internal error (for example, the SOAP XML
 147 namespace is incorrect).

148 In the case of a Liberty processing error, the SOAP HTTP server MUST respond with "200 OK" and include a profile-
 149 specified response as the only child of the <SOAP-ENV:Body> element.

150 2.1.2.6. Example of Message Exchange Using SOAP over HTTP

151 The following is an example of the SOAP exchange for the single sign-on browser artifact profile requesting an
 152 authentication assertion (the left margin white space added for legibility invalidates the signature).

```

153
154 POST /authn HTTP/1.1
155 Host: idp.example.com
156 Content-type: text/xml
157 Content-length: nnnn
158 <soap-env:Envelope
159     xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
160     <soap-env:Header/>
161     <soap-env:Body>
162         <samlp:Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol"
163             xmlns:lib="urn:liberty:iff:2003-08"
164             xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
165             xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
166             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
167             IssueInstant="2002-12-12T10:08:56Z"
168             MajorVersion="1"
169             MinorVersion="1"
170             RequestID="e4d71c43-c89a-426b-853e-a2b0c14a5ed8"
171             id="ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1">
172             <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
173                 <ds:SignedInfo>
174                     <ds:CanonicalizationMethod
175                         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
176                     </ds:CanonicalizationMethod>
177                     <ds:SignatureMethod
178                         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
179                     </ds:SignatureMethod>
180                     <ds:Reference URI="#ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1">
181                         <ds:Transforms>
182                             <ds:Transform
183                                 Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
184                             </ds:Transform>
185                             <ds:Transform
186                                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
187                             </ds:Transform>
188                         </ds:Transforms>
189                         <ds:DigestMethod
190                             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
191                         </ds:DigestMethod>
192                         <ds:DigestValue>+k6TnolGkIPKZlPQUVyok8dwkuE=</ds:DigestValue>
193                     </ds:Reference>
194                 </ds:SignedInfo>
195                 <ds:SignatureValue>
196                     wXJMVoP01V1jFnWJPY0WqP5Gqm8A1+/2b5gNzF4L4LMu4yEcr tttLdPPT3bvhwkwxHjL9NuOFumQ
197                     5YEyiVz1NcjAxx0LfgwutvEdJb748IU4L+8obXPXfgTZLiBK1RbHCRmRvj1PIu22oGCV6Ewu iWRv
198                     OD6Ox9svtSgFJ+iXkZQ
199                 </ds:SignatureValue>

```

```

200     <ds:KeyInfo>
201       <ds:X509Data>
202         <ds:X509Certificate>
203           MIIDMTCCAppggAwIBAgIBHDANBgkqhkiG9w0BAQQFADCB1TELMAKGA1UEBhMCVVMxMzZAJBgNVBACt
204           ALN GMRkwFwYDVQQKEExBMAWJlcnR5IEFsbGhbmNLMRQwEgYDVQQLEwJTT1AgVGVzZdGVycyEiMCAG
205           AlUEAxMzTGliZXJ0eSBUZXN0ZXJzIENlcnRpbmllcjkEKMCIgCSqGSIsb3DQeJARYVcnJvZHZHJpZ3Vl
206           ekBuzW9zb2wubmV0MB4XDTAyMTIwNDE1NTg0NFoXDTEyMTIwMTE1NTg0NFowgasx CzAJBgNVBAYT
207           AlVTMQswCQYDVQQHEwJTRjEKMCIgA1UEChMhMTIwNDE1NTg0NFoXDTEyMTIwMTE1NTg0NFowgasx CzAJBgNVBAYT
208           JAYDVQQLExlJTT1AgVGVzZdGVycyBlcmljc3Nvb1hIHNPZ251cjkEXMBUAG1UEAxMOZXXJpY3Nzb24t
209           YS5pb3AxKDAmbGkqhkiG9w0BCQEWGjyb2RyaWd1ZXpAZXJpY3Nzb24tYS5pb3AwgZ8wDQYJKoZI
210           hv cNAQEBAQADgY0AMIGJAoGBAPUoGYvJxQc5jzDnJ14TV6TaTb3fH95ju24Z0y6HQxm6gXdJSAo
211           Wh7/AIes4UcV09DC2kKS6Vow2YoXt2LIyH9HWH2tEUT1js/P UeBHEWcW3tFezM6jh5GG5rCuVPZa
212           W9e oGUBFPszOPFKUAWdHUXSDWufY1KZ93 Ixh0BeZgg6VAgMBAAGjeTB3MEoGCWC GSAGG+EIBDQO9
213           FjtUaGlzIHNPZ25pbm cgY2VydCB3YXMyY3JlYXRlZCBmb3IgdGVzdGluZy4gRG8gYm90IHRydXN0
214           IGl0LjAJBgNVHRMEA jAAMBEGCWC GSAGG+ EIBAQQEAWIEMDALBgNVHQ8EBAMCBsAwDQYJKoZIhvcN
215           AQEEBQADgYEAR/HSgBpAprQwQVvYwDE9pCaiduKv4/W/+hrdpX1VKSr6TIlg4ouDCQJNos7tNuG9Z
216           AbfWtHvC5s51n2cfAzfns/DKqXRqcsxzL5ZUBksPpmsDobooopUv6Xm8RFsi7yB9AGaVuqObeY/+m
217           70nOu030+FlMN3Ulk2E3rOKX1UlnoC0</ds:X509Certificate>
218       </ds:X509Data>
219     </ds:KeyInfo>
220   </ds:Signature>
221   <samlp:AssertionArtifact>
222     AAMluXw6+f+jyA/4XuFHqPl7QDvc/LIQL9+t7YQtG1Gwk9bph0Adl+o+
223   </samlp:AssertionArtifact>
224 </samlp:Request>
225 </soap-env:Body>
226 </soap-env:Envelope>
    
```

227 The following is an example of a response, which supplies an assertion containing an authentication statement:

```

228
229 HTTP/1.1 200 OK
230 Content-Type: text/xml
231 Content-Length: nnnn
232 <soap-env:Envelope
233   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
234   <soap-env:Header/>
235   <soap-env:Body>
236     <samlp:Response
237       xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
238       InResponseTo="RPCUk211+GVz+t1lLURp51oFvJXk"
239       IssueInstant="2002-10-31T21:42:13Z" MajorVersion="1" MinorVersion="1"
240       Recipient="http://localhost:8080/sp"
241       ResponseID="LANWFL2xLybnc+BCwgY+p1/vIVAj">
242     <samlp:Status>
243       <samlp:StatusCode
244         xmlns:qns="urn:oasis:names:tc:SAML:1.0:protocol"
245         Value="qns:Success">
246       </samlp:StatusCode>
247     </samlp:Status>
248     <saml:Assertion
249       xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
250       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
251       xmlns:lib="urn:liberty:iff:2003-08"
252       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
253       AssertionID="SqMC8Hs2vJ7Z+t4UiLSmhKOSU00U"
254       InResponseTo="RPCUk211+GVz+t1lLURp51oFvJXk"
255       IssueInstant="2002-10-31T21:42:13Z" Issuer="http://localhost:8080/idp"
256       MajorVersion="1" MinorVersion="2"
257       xsi:type="lib:AssertionType">
258     <saml:Conditions
259       NotBefore="2002-10-31T21:42:12Z"
260       NotOnOrAfter="2002-10-31T21:42:43Z">
261     <saml:AudienceRestrictionCondition>
262       <saml:Audience>http://localhost:8080/sp</saml:Audience>
263     </saml:AudienceRestrictionCondition>
264     </saml:Conditions>
    
```



```

265     <saml:AuthenticationStatement
266         AuthenticationInstant="2002-10-31T21:42:13Z"
267         AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
268         xsi:type="lib:AuthenticationStatementType">
269     <saml:Subject xsi:type="lib:SubjectType">
270         <saml:NameIdentifier Format="urn:liberty:iff:nameid:federated">
271             C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
272         </saml:NameIdentifier>
273         <saml:SubjectConfirmation>
274             <saml:ConfirmationMethod>
275                 urn:oasis:names:tc:SAML:1.0:cm:artifact-01
276             </saml:ConfirmationMethod>
277         </saml:SubjectConfirmation>
278         <lib:IDPProvidedNameIdentifier>
279             C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
280         </lib:IDPProvidedNameIdentifier>
281     </saml:Subject>
282 </saml:AuthenticationStatement>
283 <ds:Signature>
284     <ds:SignedInfo>
285         <ds:CanonicalizationMethod
286             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
287         </ds:CanonicalizationMethod>
288         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
289         </ds:SignatureMethod>
290         <ds:Reference URI="">
291             <ds:Transforms>
292                 <ds:Transform
293                     Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
294                 </ds:Transform>
295                 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
296                 </ds:Transform>
297             </ds:Transforms>
298             <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
299             </ds:DigestMethod>
300             <ds:DigestValue>ZbscbqHTX9H8bBftRIWlG4Epv1A= </ds:DigestValue>
301         </ds:Reference>
302     </ds:SignedInfo>
303     <ds:SignatureValue>
304         H+q3nC3jUaljlUKUVkcC4iTfClxeZQIFF0nvHqPS5oZhtkB aDb9qITA7gIkotaB584wXqTXwsfsu
305         IrwT5uL3r85Rj7IF6NeCeiy3K0+z3uewxyeZ Pz8wna449VNm0qNHYkgNak9ViNCp0/ ks5MAttoPo
306         2iLOfaKu3wWG6d1G+DM=
307     </ds:SignatureValue>
308 </ds:Signature>
309 </saml:Assertion>
310 </samlp:Response>
311 </soap-env:Body>
312 </soap-env:Envelope>

```

313 2.1.2.7. Example of Message Exchange Using URL-encoding

```

314
315 http://127.0.0.1:8080/tfssidp/IDPSingleSignOnServiceV12?RequestID=dd
316 d4aa20-24d4-4366-8f60-7bb0e068334a&MajorVersion=1&MinorVersion=2&IssueInstant=
317 20
318 03-07-26T05%3A44%3A00Z&ProviderID=http%3A%2F%2F127.0.0.1%3A8081%2Ftfssp&NameIDPo
319 licy=none&IsPassive=0&RelayState=3f53f0934a63728de06f555493683e131ad131fff&
320 SigAlg
321 =http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-
322 sha1&Signature=kAfiiHlyI8OJ
323 G6IGV6Yl7ToaxwF1E4ww%2FronqAkkC1lKyxvjc03%2Bitx7Q44vRVDm%2BbAm38ofFSsb%0AbzYsWgT
324 MAiDPsm2wQUuYP2oRB9AApS%2BdzWfGWhqV%2FU0malW7cRMDBA2ewyAQTDiaLLeSetQ6zAnJ%0A4KzH
325 ExQ1%2BzelPehsWoe%3D
326

```

327 Example 1. URL-encoded <AuthnRequest>

328

329 `http://127.0.0.1:8081/xfss/SPAssertionConsumerSe`

330 `viceV1.2?SAMLart=AAMZf3sIj1R%2BgQe%2BMVfP6flshoGaPG1XrZG6E%2B212nHDCxFA7h%2BDcE`

331 `%2B&RelayState=3f53f0934a63728de06f555493683e131ad131ff`

332

333 Example 2. URL-encoded <AuthnResponse>

334 3. Profiles

335 This section defines the Liberty profiles for the use of request and response messages defined in [\[LibertyProtSchema\]](#)
336 and [\[SAMLCore11\]](#). The combination of message content specification and message transport mechanisms for a
337 single client type (that is, user agent) is termed a *Liberty profile*. The profiles have been grouped into categories
338 according to the protocol message intent.

339 The following profile categories are defined in this document:

- 340 • **Single Sign-On and Federation:** The profiles by which a service provider obtains an authentication assertion
341 from an identity provider facilitating single sign-on and identity federation.
- 342 • **Name Registration:** The profiles by which service providers and identity providers specify the name identifier to
343 be used when communicating with each other about the Principal.
- 344 • **Federation Termination Notification:** The profiles by which service providers and identity providers are notified
345 of federation termination.
- 346 • **Single Logout:** The profiles by which service providers and identity providers are notified of authenticated
347 session termination.
- 348 • **Identity Provider Introduction:** The profile by which a service provider discovers which identity providers a
349 Principal may be using.
- 350 • **Name Identifier Mapping:** The profile by which a service provider may obtain a NameIdentifier with which to
351 refer to a Principal at a SAML Authority.
- 352 • **Name Identifier Encryption:** The profile by which one provider may encrypt a NameIdentifier to permit it to pass
353 through a third-party without revealing the actual value until received by the intended provider.

354 3.1. Common Requirements

355 The following rules apply to all profiles in this specification, unless otherwise noted by the individual profile.

- 356 1. All HTTP requests and responses **MUST** be drawn from either HTTP 1.1 (see [\[RFC2616\]](#)) or HTTP 1.0 (see
357 [\[RFC1945\]](#)). When an HTTP redirect is specified, the HTTP response **MUST** have a status code of "302".
358 According to HTTP 1.1 and HTTP 1.0, the use of status code 302 is recommended to indicate "the requested
359 resource resides temporarily under a different URI." The response may also include additional headers and an
360 optional message.
- 361 2. When `https` is specified as the `<scheme>` for a URL, the HTTP connection **MUST** be made over either SSL 3.0
362 (see [\[SSL\]](#)) or TLS 1.0 (see [\[RFC2246\]](#)) or any subsequent protocols that are backwards compatible with SSL 3.0
363 and/or TLS 1.0. Other security protocols **MAY** be used as long as they implement equivalent security measures.
- 364 3. Messages between providers **MUST** have their integrity protected, confidentiality **MUST** be ensured and the
365 recipient **MUST** authenticate the sender.
- 366 4. Providers **MUST** use secure transport (`https`) to achieve confidentiality and integrity protection. The initiator of
367 the secure connection **MUST** authenticate the server using server-side X.509 certificates.
- 368 5. The authenticated identity of an identity provider **MUST** be securely available to a Principal before the Principal
369 presents his/her personal authentication data to that identity provider.

- 370 6. Certificates and private keys **MUST** be suitable for long-term signatures. See [\[LibertyProtSchema\]](#) for guidelines
371 on signature verification. For signing and verification of protocol messages, identity and service providers
372 **SHOULD** use certificates and private keys that are distinct from the certificates and private keys applied for
373 SSL or TLS channel protection.
- 374 7. In transactions between service providers and identity providers, requests **MUST** be protected against replay, and
375 received responses **MUST** be checked for correct correspondence with issued requests. (**Note:** Other steps may
376 intervene between the issuance of a request and its eventual response within a multistep transaction involving
377 redirections.) Additionally, time-based assurance of freshness **MAY** be provided.
- 378 8. Each service provider within a circle of trust **MUST** be configured to enable identification of the identity providers
379 whose authentications it will accept. Each identity provider **MUST** be configured to enable identification of the
380 service providers it intends to serve.
381 **Note:**
382 The format of this configuration is a local matter and could, for example, be represented as lists of names or as
383 sets of X.509 certificates of other circle of trust members.
- 384 9. Circle of trust bilateral agreements on selecting certificate authorities, obtaining X.509 credentials, establishing
385 and managing trusted public keys, and tracking lifecycles of corresponding credentials are assumed and not in
386 scope for this specification.
- 387 10. The <scheme> of the URL for SOAP endpoints **MUST** be `https`.
- 388 11. All SOAP message exchanges **MUST** adhere to the SOAP protocol binding for Liberty (see [Section 2.1](#)).

389 3.1.1. User Agent

390 Unless otherwise noted in the specific profile, a user agent **MUST** support the following features to be interoperable
391 with the protocols in [\[LibertyProtSchema\]](#) and Liberty profiles in this document:

- 392 • HTTP 1.0 (see [\[RFC1945\]](#)) or HTTP 1.1 (see [\[RFC2616\]](#)).
- 393 • SSL 3.0 (see [\[SSL\]](#)) or TLS 1.0 (see [\[RFC2246\]](#)) or any subsequent protocols which are backwards compatible
394 with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for example, a WAP gateway).
- 395 • Minimum maximum URL length of 256 bytes. See [\[LibertyGlossary\]](#) for definition.
- 396 • A WAP browser user agent **MUST** support WML 1.0,1.1, 1.2 or 1.3 [\[WML\]](#) in addition to the above requirements.

397 Additionally, to support the optional identity provider introduction profile, either the user agent or a proxy must support
398 session cookies (see [\[RFC2109\]](#)). Support for persistent cookies will yield a more seamless user experience.

399 **3.1.2. Formatting and Encoding of Protocol Messages**

400 All protocol messages that are indicated by the profile as being communicated in the `<query>` component of the URL
401 MUST adhere to the formatting and encoding rules in [Section 3.1.2.1](#).

402 **3.1.2.1. Encoding URL-embedded Messages**

403 URL-embedded messages are encoded using the `application/x-www-form-urlencoded` MIME type as if they
404 were generated from HTML forms with the GET method as defined in [\[HTML4\]](#).

405 The original XML protocol message MUST be encoded as follows:

406 • The `<query>` component parameter value MUST be the value of the XML protocol message element or attribute
407 value.

408 • The value of the `<query>` component parameter MUST be a space-delimited list when the original message
409 element has multiple values.

410 • Some of the referenced protocol message elements and attributes are optional. If an optional element or attribute
411 does not appear in the original XML protocol message, then the corresponding data item MUST be omitted from
412 the URL encoded message.

413 • URLs appearing in the URL-encoded message SHOULD NOT exceed 80 bytes in length (including %-escaping
414 overhead). Likewise, the `<lib:RelayState>` data value SHOULD NOT exceed 80 bytes in length.

415 • The URL-encoding of status codes in the responses `RegisterNameIdentifierResponse` and
416 `LogoutResponse` may be taken from several sources. The top level codes MUST be from SAML.
417 Other codes (including Liberty-defined values) MAY be used at the second or lower levels. The URL parameter
418 value should be interpreted as a QName with the "lib", "saml", and "samlp" namespaces pre-defined to their
419 respective namespace URIs. Query parameters with the name "xmlns:prefix" can be used to map additional
420 namespace prefixes for the purpose of QName resolution, so long as the `xmlns:prefix` URL parameter appears
421 before the URL parameter containing the QName which needs the prefix definition.

422 As `<samlp:StatusCode>` elements may be nested hierarchically (see [\[\[SAMLCore11\]](#)), there may exist
423 multiple values for `<samlp:StatusCode>` in the response messages. These multiple values MUST be encoded by
424 producing a URL-encoded space-separated string as the value of this query parameter. An example is as follows:

425 `Value=samlp%3AResponder%20lib%3AFederationDoesNotExist`
426
427

428 • Certain XML protocol messages support extensibility via an `<Extension>` element. Messages that are to be
429 URL-encoded MUST adhere to the following restrictions when including extension content:
430

431 • Only attribute values and elements with simple content models are permitted.

432 • All attributes and elements MUST have an empty namespace and MUST have unique local names.

433 • Each value included SHOULD NOT exceed 80 bytes in length (including encoding overhead).

434 XML digital signatures are not directly URL-encoded due to space concerns. If the Liberty XML protocol message is
435 signed with an XML signature, the encoded URL form of the message MUST be signed as follows:

- 436 • Include the signature algorithm identifier as a new <query> component parameter named SigAlg, but omit the
437 signature.
 - 438 • Sign the string containing the URL-encoded message. The string to be signed MUST include only the <query>
439 part of the URL (that is, everything after ? and before &Signature=). Any required URL-escaping MUST be
440 done before signing.
 - 441 • Encode the signature using base64 (see [\[RFC2045\]](#)).
 - 442 • Add the base64-encoded signature to the encoded message as a new data item named Signature.
- 443 Note that some characters in the base64-encoded signature value may require URL escaping before insertion into the
444 URL <query> part, as is the case for any other data item value.
- 445 Any items added after the Signature <query> component parameter are implicitly unsigned.
- 446 The service URL provided by the provider (the URL to which <query> parameters are added) MUST NOT contain
447 any pre-existing <query> parameter values.
- 448 The following signature algorithms (i.e., DSAwithSHA1, RSAwithSHA1) and their identifiers (the URIs) MUST be
449 supported:
- 450 • DSAwithSHA1 - <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
 - 451 • RSAwithSHA1 - <http://www.w3.org/2000/09/xmldsig#rsa-sha1>

452 3.1.2.1.1. Size Limitations

453 When the request initiator knows or suspects that the user agent cannot process the full URL-encoded message in the
454 URL due to size considerations, the requestor MAY send the Liberty XML protocol message using a form POST.
455 The form MUST be constructed with contents that contain the field LAREQ or LARES with the respective value being
456 the Liberty XML protocol request or response message (e.g., <lib:AuthnRequest> or <lib:AuthnResponse>)
457 as defined in [\[LibertyProtSchema\]](#). The Liberty XML protocol message MUST be encoded by applying a base64
458 transformation (refer to [\[RFC2045\]](#)) to the XML message and all its elements.

459 3.1.2.1.2. URL-encoded <lib:AuthnRequest>

460 The original <lib:AuthnRequest> message:

```
461
462     <lib:AuthnRequest RequestID="[RequestID]"
463       MajorVersion="[MajorVersion]"
464       MinorVersion="[MinorVersion]"
465       IssueInstant="[IssueInstant]">
466       <lib:ProviderID>[ProviderID]</lib:ProviderID>
467       <lib:AffiliationID>[AffiliationID]</lib:AffiliationID>
468       <lib:ForceAuthn>[ForceAuthn]</lib:ForceAuthn>
469       <lib:IsPassive>[IsPassive]</lib:IsPassive>
470       <lib:NameIDPolicy>[NameIDPolicy]</lib:NameIDPolicy>
471       <lib:ProtocolProfile>[ProtocolProfile]</lib:ProtocolProfile>
472       <lib:AssertionConsumerServiceID>[AssertionConsumerServiceID]</lib:AssertionConsu
473 merServiceID>
474       <lib:AuthnContext>
475         <lib:AuthnContextStatementRef>[AuthnContextStatementRef]</lib:AuthnCon
476 textStatementRef>
477       </lib:AuthnContext>
478       <lib:RelayState>[RelayState]</lib:RelayState>
479       <lib:AuthnContextComparison>[AuthnContextComparison]</lib:AuthnContextComparison>
480       <lib:Scoping>
```

```
481         <lib:ProxyCount>[ProxyCount]</lib:ProxyCount>
482     </lib:Scoping>
483 </lib:AuthnRequest>
```

- 484 • Data elements that MUST be included in the encoded data with their values as indicated in brackets above if
485 present in the original message:

```
486
487 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID, AffiliationID, ForceAuthn,
488 IsPassive, NameIDPolicy, ProtocolProfile, AuthnContextStatementRef, AuthnContextClassRef,
489 AuthnContextComparison, RelayState, ProxyCount.
```

- 490 • Example of <lib:AuthnRequest> message URL-encoded and signed:

```
491
492 http://idp.example.com/authn?RequestID=RMvY34pg%2FV9aGJ5yw0HL0AcjcqQF
493 &MajorVersion=1&MinorVersion=2&IssueInstant=2002-05-15T00:3A58%3A19
494 &ProviderID=http%3A%2F%2Fsp.example.com%2FLiberty%2F&ForceAuthn=true
495 &IsPassive=false&NameIDPolicy=federated&ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fp
496 rofiles%2Fbrws-post
497 http%3A%2F%2Fwww.projectliberty.org%2Fschemas%2Fauthctx%2Fclasses%2FPasswordProtectedTransport
498
499 &RelayState=03mhakSms5tmQ0WRDCEzpf7BNcywZa75FwIcSSEPvbkofxaQHCuNnc5yChId
500 DlWc7JBV9Xbw3avRBK7VFsPl2X
501 &SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%2F3rsa-sha1
502 &Signature=EoD8bNr2jEOe%2Fumon6oU%2FZGIIIF7gbJAe4MLUUMrD%2BPP7P8Yf3gfdZG2qPJdNAJkzVHGfO8W8DzpQ
503 %0D%0AsDTTd5VP9MLPcvxbFQoF0CJJmvL26cPsuc54q7ourcH0jJ%2F2Ukdq4DA1Y1Z5kPIg%2Bt rykgLz0U%2BS%0D%0ANqpNHkjh6W3Y
504 kGv7RBs%3D
```

505 3.1.2.1.3. URL-Encoded <lib:FederationTerminationNotification>

506 The original <lib:FederationTerminationNotification> message:

```
507
508 <lib:FederationTerminationNotification ...
509     RequestID="[RequestID]"
510     MajorVersion="[MajorVersion]"
511     MinorVersion="[MinorVersion]"
512     IssueInstant="[IssueInstant]">
513     <lib:ProviderID>[ProviderID]</lib:ProviderID>
514     <saml:NameIdentifier
515         NameQualifier="[NameQualifier]"
516         Format="[NameFormat]">[NameIdentifier]</saml:NameIdentifier>
517 </lib:FederationTerminationNotification>
518
```

- 519 • Data elements that MUST be included in the encoded data with their values as indicated in brackets above if
520 present in the original message:

```
521
522 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID, NameQualifier, NameFormat,
523 NameIdentifier.
```

524 3.1.2.1.4. URL-Encoded <lib:LogoutRequest>

525 The original <lib:LogoutRequest> message:

```
526
527 <lib:LogoutRequest ...
528   RequestID="[RequestID]"
529   MajorVersion="[MajorVersion]"
530   MinorVersion="[MinorVersion]"
531   IssueInstant="[IssueInstant]">
532   <lib:ProviderID>[ProviderID]</lib:ProviderID>
533   <saml:NameIdentifier
534     NameQualifier="[NameQualifier]"
535     Format="[NameFormat]">
536     [NameIdentifier]
537   </saml:NameIdentifier>
538   <lib:SessionIndex>[SessionIndex]</lib:SessionIndex>
539   <lib:RelayState>[RelayState]</lib:RelayState>
540 </lib:LogoutRequest>
```

541 • Data elements that MUST be included in the encoded data with their values as indicated in brackets above if
542 present in the original message:

```
543
544 RequestID, MajorVersion, MinorVersion, IssueInstant,
545 ProviderID, NameQualifier, NameFormat, NameIdentifier,
546 SessionIndex, RelayState.
```

547 3.1.2.1.5. URL-Encoded <lib:LogoutResponse>

548 The <lib:LogoutResponse> response message:

```
549
550 <lib:LogoutResponse
551   ResponseID="[ResponseID]"
552   InResponseTo="[InResponseTo]"
553   MajorVersion="[MajorVersion]"
554   MinorVersion="[MinorVersion]"
555   IssueInstant="[IssueInstant]"
556   Recipient="[Recipient]">
557 <lib:ProviderID>[ProviderID]</lib:ProviderID>
558 <samlp:Status>
559 <samlp:StatusCode Value="[Value]"/>
560 </samlp:Status>
561 <lib:RelayState>[RelayState]</lib:RelayState>
562 </lib:LogoutResponse>
```

563 • Data elements that MUST be included in the encoded data with their values as indicated in brackets above if
564 present in the original message:

```
565
566 ResponseID, InResponseTo, MajorVersion, MinorVersion, IssueInstant, Recipient, ProviderID,
567 Value, RelayState.
```

568 • The <lib:LogoutResponse> message may contain nested status code information. Multiple values MUST be
569 URL-encoded by creating a space-separated list (see general requirements at top of [Section 3.1.2.1.5](#)).

570 3.1.2.1.6. URL-Encoded <lib:RegisterNameIdentifierRequest>

571 The original <lib:RegisterNameIdentifierRequest> message:

```
572
573 <lib:RegisterNameIdentifierRequest
574   RequestID="[RequestID]"
575   MajorVersion="[MajorVersion]"
576   MinorVersion="[MinorVersion]"
577   IssueInstant="[IssueInstant]">
578 <lib:ProviderID>[ProviderID]</lib:ProviderID>
579 <lib:IDPProvidedNameIdentifier
580   NameQualifier="[IDPNameQualifier]"
581   Format="[IDPNameFormat]">[IDPProvidedNameIdentifier]
582 </lib:IDPProvidedNameIdentifier>
583 <lib:SPPProvidedNameIdentifier
584   NameQualifier="[SPNameQualifier]"
585   Format="[SPNameFormat]">[SPPProvidedNameIdentifier]
586 </lib:SPPProvidedNameIdentifier>
587 <lib:OldProvidedNameIdentifier
588   NameQualifier="[OldNameQualifier]"
589   Format="[OldNameFormat]">[OldProvidedNameIdentifier]
590 </lib:OldProvidedNameIdentifier>
591 <lib:RelayState>[RelayState]</lib:RelayState>
592 </lib:RegisterNameIdentifierRequest>
593
```

594 • Data elements that MUST be included in the encoded data with their values as indicated in brackets above if
595 present in the original message:

```
596
597   RequestID, MajorVersion, MinorVersion, IssueInstant,
598   ProviderID, IDPNameQualifier, IDPNameFormat, IDPProvidedNameIdentifier,
599   SPNameQualifier, SPNameFormat, SPPProvidedNameIdentifier,
600   OldNameQualifier, OldNameFormat, OldProvidedNameIdentifier,
601   RelayState
```

602 3.1.2.1.7. URL-Encoded <lib:RegisterNameIdentifierResponse>

603 The <lib:RegisterNameIdentifierResponse> response message:

```
604
605 <lib:RegisterNameIdentifierResponse
606   ResponseID="[ResponseID]"
607   InResponseTo="[InResponseTo]"
608   MajorVersion="[MajorVersion]"
609   MinorVersion="[MinorVersion]"
610   IssueInstant="[IssueInstant]"
611   Recipient="[Recipient]">
612 <lib:ProviderID>[ProviderID]</lib:ProviderID>
613 <samlp:Status>
614   <samlp:StatusCode Value="[Value]"/>
615 </samlp:Status>
616 <lib:RelayState>[RelayState]</lib:RelayState>
617 </lib:RegisterNameIdentifierResponse>
618
```

619 • Data elements that MUST be included in the encoded data with their values as indicated in brackets above if
620 present in the original message:

```
621
622   ResponseID, InResponseTo, MajorVersion, MinorVersion,
623   IssueInstant, Recipient, ProviderID, Value, RelayState
624
```

- 625 • The `<lib:RegisterNameIdentifierResponse>` message may contain nested status code information. Multiple values MUST be URL-encoded by creating a space-separated list (see general requirements at top of
626 Section 3.1.2.1.
627

628 3.1.3. Provider Metadata

629 The majority of the Liberty profiles defined in this document rely on metadata that specify the policies that govern
630 the behavior of the service provider or identity provider. These provider metadata may be shared out of band between
631 an identity provider and a service provider prior to the exchange of Liberty protocol messages or with the protocols
632 described in [\[LibertyMetadata\]](#). The provider metadata relevant to each profile are listed in this document at the
633 beginning of the profile category. Refer to [\[LibertyMetadata\]](#) for a complete enumeration of the Liberty provider
634 metadata elements and their associated schema.

635 3.2. Single Sign-On and Federation Profiles

636 This section defines the profiles by which a service provider obtains an authentication assertion of a user agent from
637 an identity provider to facilitate single sign-on. Additionally, the single sign-on profiles can be used as a means of
638 federating an identity from a service provider to an identity provider through the use of the `<NameIDPolicy>` element
639 in the `<lib:AuthnRequest>` protocol message as specified in [\[LibertyProtSchema\]](#).

640 The single sign-on profiles make use of the following metadata elements, as defined in [\[LibertyProtSchema\]](#):

- 641 • `ProviderID` Used to uniquely identify the service provider to the identity provider and is documented in these
642 profiles as "service provider ID."
- 643 • `AffiliationID` Used to uniquely identify an affiliation group to the identity provider and is documented in
644 these profiles as "affiliation ID."
- 645 • `SingleSignOnServiceURL` The URL at the identity provider that the service provider should use when sending
646 single sign-on and federation requests. It is documented in these profiles as "single sign-on service URL."
- 647 • `AssertionConsumerServiceURL` The URL(s) at the service provider that an identity provider should use when
648 sending single sign-on or federation responses. It is documented in these profiles as "assertion consumer service
649 URL."
- 650 • `SOAPEndpoint` The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP
651 messages are sent.

652 3.2.1. Common Interactions and Processing Rules

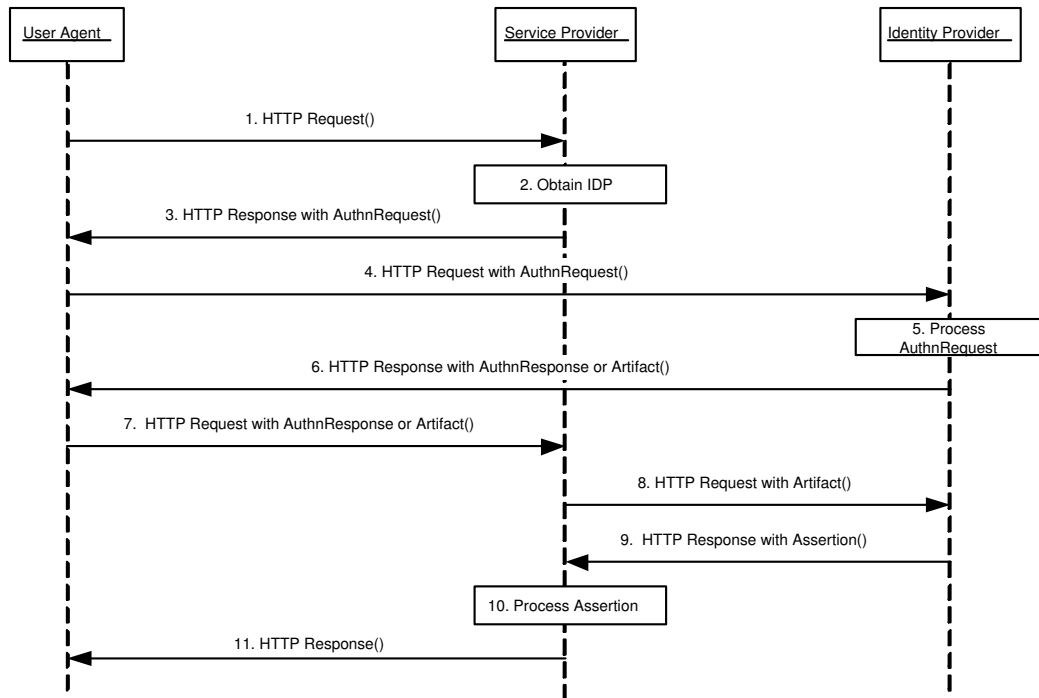
653 This section defines the set of interactions and process rules that are common to all single sign-on profiles.

654 All single sign-on profiles can be described by one interaction diagram, provided that different messages are optional
 655 in different profiles and that the actual content of the messages may differ slightly. Where interactions and messages
 656 differ or are optional, they are designated and detailed within the specific single sign-on profiles. [Figure 1](#) represents
 657 the basic template of interactions for achieving single sign-on. This should be used as the baseline for all single sign-on
 658 profiles.

659 It should be noted that multiple identity providers may be involved in the authentication of the Principal. Although
 660 a single identity provider is depicted in the profiles below ([Figure 1](#), that identity provider *MAY* interact with other
 661 identity providers to authenticate the Principal using the proxying method described in [\[LibertyProtSchema\]](#) and the
 662 profiles as noted below. In such situations these profiles would be used by the identity provider originally contacted
 663 by the requesting service provider to communicate with additional identity providers.

664 In the figure below, steps 1 through 5 can be considered typical but optional. An identity provider *MAY* initiate a
 665 SSO profile by unilaterally creating a `<lib:AuthnResponse>` or artifact, and proceeding with step 6, as discussed
 666 in [\[LibertyProtSchema\]](#).

667 Step 4 specifies that an HTTP request containing the `<lib:AuthnRequest>` is sent to the identity provider. Step
 668 7 is comparable, containing an artifact or `<lib:AuthnResponse>`. This request may be a GET or POST request;
 669 providers *MUST* support both methods. As described in [Section 3.1.2.1.1](#), such a POST *MUST* contain an `LAREQ` or
 670 `LARES` form element containing the XML protocol request or response (or artifact), respectively, in base64-encoded
 671 format.



672

673 [Figure 1](#). Basic single sign-on profile.

674 3.2.2. Liberty Artifact Profile

675 The Liberty artifact profile relies on a reference to the needed assertion traveling in a SAML artifact, which the service
 676 provider must dereference from the identity provider to determine whether the Principal is authenticated. This profile
 677 is an adaptation of the "Browser/artifact profile" for SAML as documented in [\[SAMLBind11\]](#). See [Figure 3](#).

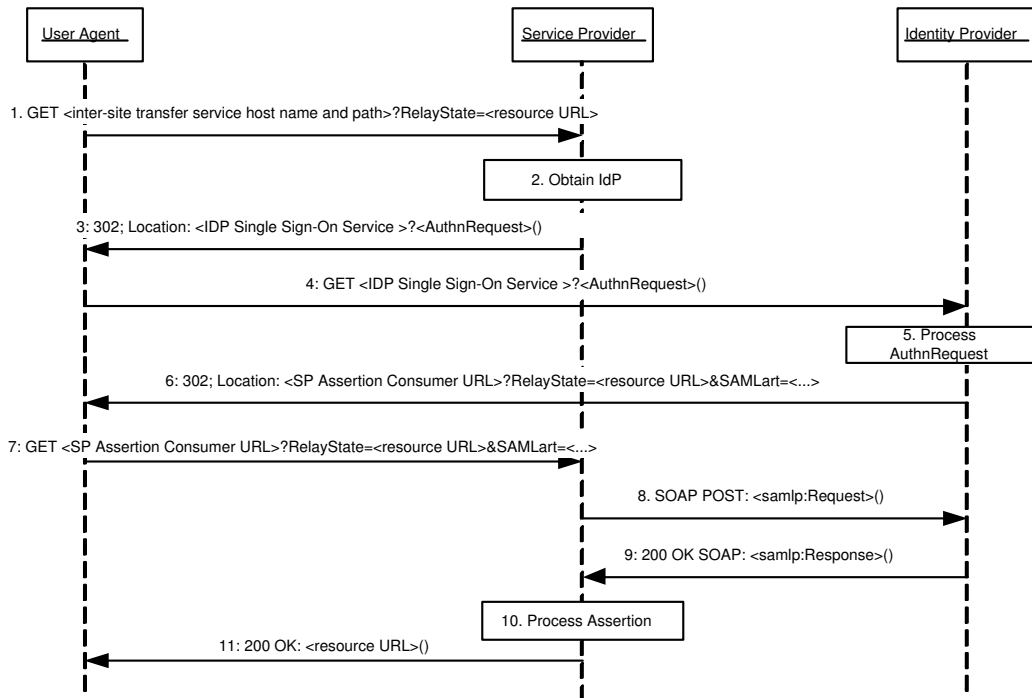
678 The following URI-based identifier **MUST** be used when referencing this specific profile (for example,
679 <lib:ProtocolProfile> element of the <lib:AuthnRequest> message):

680 URI: `http://projectliberty.org/profiles/brws-art`

681 The Liberty artifact profile consists of a single interaction among three parties: a user agent, an identity provider, and
682 a service provider, with a nested subinteraction between the identity provider and the service provider.

683 3.2.2.1. Interactions

684 **Figure 2** illustrates the Liberty artifact profile for single sign-on.



685

686

Figure 2. Liberty artifact profile for single sign-on

687 This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1.
688 Thus, a valid session exists for the user agent at the identity provider. When implementing this profile, all processing
689 rules defined in [Section 3.2.1](#) for the single sign-on profiles **MUST** be followed. Additionally, the following rules
690 **MUST** be observed as they relate to steps 3, 6 and 7:

691 3.2.2.1.1. Step 3: Single sign on Service with <AuthnRequest>

692 In step 3, the service provider’s intersite transfer service responds and instructs the user agent to access the single
693 sign-on service URL at the identity provider.

694 This step may take place via an HTTP 302 redirect, a WML redirect deck or any other method that results in the user
695 agent being instructed to make an HTTP GET or POST request to the identity provider’s single signon service.

696 This response **MUST** adhere to the following rules:

- 697 • The response **MUST** contain the identity provider’s single sign-on service URL (for example, as the Location
698 header of an HTTP 302 redirect, the `action` attribute of an `HTMLForm` or the `href` attribute of a `<go>` element
699 in a WML redirect deck).

- 700 • The identity provider's single sign-on service URL MUST specify `https` as the URL scheme.
701 **Note:**
702 Future protocols may be adopted and enabled to work within this framework. Therefore, implementers are
703 encouraged to not hardcode a reliance on `https`.
- 704 • The response MUST include one of the following:
705
- 706 • A `<query>` component containing the `<lib:AuthnRequest>` protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2.
707
708 **Note:**
709 The `<lib:RelayState>` element of the `<lib:AuthnRequest>` message can be used by the service provider
710 to help maintain state information during the single sign-on and federation process. For example, the originally
711 requested resource (i.e., `RelayState` in step 1) could be stored as the value for the `<lib:RelayState>` element,
712 which would then be returned to the service provider in the `<lib:AuthnResponse>` in step 7. The service
713 provider could then use this information to formulate the HTTP response to the user agent in step 11.
- 714 • An HTTP form containing the field `LAREQ` with the value of the `<lib:AuthnRequest>` protocol message
715 as defined in [LibertyProtSchema]. The `<lib:AuthnRequest>` MUST be encoded by applying a base64
716 transformation (see [RFC2045]).
- 717 Implementation examples:

- 718 • HTTP 302 Redirect
719
720
721 `<HTTP-Version> 302 <Reason Phrase>`
722 `<other headers>`
723 `Location: https://<Identity Provider Single Sign-On Service host name and path>?<query>`
724 `<other HTTP 1.0 or 1.1 components>`
725
726
- 727 • HTML Form POST
728
729
730 `<html>`
731 `<body onLoad="document.forms[0].submit()">`
732 `<form action="https://<Identity Provider Single Sign-On Service host name and path">`
733 `method="POST" >`
734 `<input type="hidden" name="LAREQ" value="<base 64 encoded AuthnRequest>" >`
735 `</form>`
736 `</body>`
737 `</html>`
738
739

740 • WML Redirect with POST

```

741           ...
742
743     <wml>
744     <card id="redirect" title="Log In">
745       <onenterforward>
746         <go method="post" href="<Identity Provider Single Sign-On service host name and path>" >
747           <postfield name="LAREQ" Value="<base64-encoded AuthnRequest>" />
748         </go>
749       </onenterforward>
750       <onenterbackward>
751         <prev/>
752       </onenterbackward>
753       <p>
754         Contacting IdP. Please wait...
755       </p>
756       ...
757     </card>
758     ...
759   </wml>
760
761

```

762 • WML Redirect with GET

```

763           ...
764
765     <wml>
766     <card id="redirect" title="Log In">
767       <onenterforward>
768         <go href="<Identity Provider Single Sign-On service host name and path>?<query>" />
769       </onenterforward>
770       <onenterbackward>
771         <prev/>
772       </onenterbackward>
773       <p>
774         Contacting IdP. Please wait...
775       </p>
776       ...
777     </card>
778     ...
779   </wml>
780
781

```

782 where:

783 <Identity Provider Single Sign-On service host name and path>

784 This element provides the host name, port number, and path components of the single sign-on service URL at the
785 identity provider.

786 <query>= ...<URL-encoded AuthnRequest> ...

787 A <query> component MUST contain a single authentication request:

788 <base64-encoded AuthnRequest>

789 A <base64-encoded AuthnRequest> component MUST contain a single authentication request message in
790 base64-encoded form.

791 **3.2.2.1.2. Step 6: Redirecting to the Service Provider**

792 In step 6, the identity provider instructs the user agent to access the the service provider's assertion consumer service
793 URL, and provides a SAML artifact for de-referencing by the service provider.

794 This step may take place via an HTTP 302 redirect, a WML redirect deck or any other method that results in the user
795 agent being instructed to make an HTTP GET or POST request to the service provider's assertion consumer service.

796 This response MUST adhere to the following rules:

797 • The response MUST contain the service provider's assertion consumer service URL (for example, as the Location
798 header of an HTTP 302 redirect, the `action` attribute of an `HTMLForm` or the `href` attribute of a `<go>` element
799 in a WML redirect deck).

800 • The service provider's assertion consumer service URL MUST specify `https` as the URL scheme.

801 **Note:**

802 Future protocols may be adopted and enabled to work within this framework. Therefore, implementers are
803 encouraged to not hardcode a reliance on `https`.

804 • The response MUST include one of the following:

805

806 • A `<query>` component containing a parameter `SAMLart`, the value of which is the SAML artifact on success
807 or on failure. In the case of failure, the status will be conveyed in the `<saml:Response>` returned in Step 9.
808 Additionally, if the `<lib:AuthnRequest>` processed in Step 5 included a value for the `<lib:RelayState>`
809 element, then a parameter named `RelayState` with a value set to that of the `<lib:RelayState>` element MUST
810 be included in the `<query>` component.

811 • An HTTP form containing the field `LARES` with the value of the SAML Artifact as defined in [Section 3.2.2.2](#)
812 If a value for `<RelayState>` was supplied in the `<lib:AuthnRequest>`, then the form MUST contain a
813 field `RelayState`, with a value obtained from that element in the `<lib:AuthnRequest>`.

814 • All SAML artifacts returned MUST contain the same identity provider ID.

815 Implementation examples:

816 • HTTP 302 Redirect

817

818

819 `<HTTP-Version> 302 <Reason Phrase>`

820 `<other headers>`

821 `Location: https://<Service Provider Assertion Consumer Service host name and path>?<query>`

822 `<other HTTP 1.0 or 1.1 components>`

823

824

825 • HTML Form POST

826

827

828 `<html>`

829 `<body onLoad="document.forms[0].submit()">`

830 `<form action="https://<Service Provider Assertion Consumer Service host name and path>"`

831 `method="POST">`

832 `<input type="hidden" name="LAREQ" value="<SAML Artifact>" >`

833 `<input type="hidden" name="RelayState" value="<RelayState>" >`

834 `</form>`

835 `</body>`

836 `</html>`

837

838

839 • WML Redirect with POST

```
840
841     ...
842     <wml>
843     <card id="redirect" title="Artifact">
844     <onenterforward>
845     <go method="post" href="<Service Provider Assertion Consumer Service host name and path>" >
846     <postfield name="LARES" Value="<SAML Artifact>" />
847     <postfield name="RelayState" Value="<RelayState>" />
848     </go>
849     </onenterforward>
850     <onenterbackward>
851     <prev/>
852     </onenterbackward>
853     <p>
854     Contacting IdP. Please wait...
855     </p>
856     ...
857     </card>
858     ...
859     </wml>
860
861
```

862 • WML Redirect with GET

```
863
864     ...
865     <wml>
866     <card id="redirect" title="Artifact">
867     <onenterforward>
868     <go href="<Service Provider Assertion Consumer Service host name and path>?<query>" />
869     </onenterforward>
870     <onenterbackward>
871     <prev />
872     </onenterbackward>
873     <p>
874     Contacting IdP. Please wait...
875     </p>
876     ...
877     </card>
878     ...
879     </wml>
880
881
```

882 where:

883 <Service Provider Assertion Consumer Service host name and path>

884 This element provides the host name, port number, and path components of the assertion consumer service URL at the
885 service provider.

886 <query>= ...SAMLArt=<SAML Artifact> ...RelayState=<resource URL>

887 A <query> component MUST contain at least one SAML Artifact. A single RelayState MUST be included if a value
888 for the <RelayState> was provided in the <lib:AuthnRequest>. All SAML Artifacts included MUST contain the
889 same identity provider ID (see Section 3.2.2.2).

890 <SAML Artifact>

891 A <SAML Artifact> component MUST contain at least one SAML Artifact.

892 <RelayState>

893 A form field named RelayState, with the value of that element from the <lib:AuthnRequest> MUST be included
894 if a value for the <RelayState> was provided in the <lib:AuthnRequest> and the HTTP request is made using a
895 POST.

896 3.2.2.1.3. Step 7: Accessing the Assertion Consumer Service

897 In step 7, the user agent accesses the assertion consumer service URL at the service provider, with a SAML artifact
898 representing the Principal's authentication information attached to the URL.

899 3.2.2.2. Artifact Format

900 The artifact format includes a mandatory two-byte artifact type code, as follows:

```
901  
902  
903 SAML_artifact      := B64( TypeCode RemainingArtifact )  
904 TypeCode           := Byte1Byte2  
905  
906
```

907 The notation B64(TypeCode RemainingArtifact) represents the application of the base64 transformation to the
908 catenation of the TypeCode and RemainingArtifact. This profile defines an artifact type of type code 0x0003,
909 which is REQUIRED (mandatory to implement) for any implementation of the Liberty browser artifact profile. This
910 artifact type is defined as follows:

```
911  
912  
913 TypeCode           := 0x0003  
914 RemainingArtifact := IdentityProviderSuccinctID AssertionHandle  
915 IdentityProviderSuccinctID := 20-byte_sequence  
916 AssertionHandle    := 20-byte_sequence  
917  
918
```

919 IdentityProviderSuccinctID is a 20-byte sequence used by the service provider to determine identity provider
920 identity and location. It is assumed that the service provider will maintain a table of IdentityProviderSuccinctID
921 values as well as the URL (or address) for the corresponding SAML responder at the identity provider. This
922 information is communicated between the identity provider and service provider out of band. On receiving the SAML
923 artifact, the service provider determines whether the IdentityProviderSuccinctID belongs to a known identity
924 provider and, if so, obtains the location before sending a SAML request.

925 Any two identity providers with a common service provider MUST use distinct IdentityProviderSuccinctID
926 values. Construction of AssertionHandle values is governed by the principles that the values SHOULD have no
927 predictable relationship to the contents of the referenced assertion at the identity provider, and that constructing or
928 guessing the value of a valid, outstanding assertion handle MUST be infeasible.

929 The following rules MUST be followed for the creation of SAML artifacts at identity providers:

- 930 • Each identity provider selects a single identification URL, corresponding to the provider metadata element
931 ProviderID specified in [\[LibertyMetadata\]](#).
- 932 • The identity provider constructs the IdentityProviderSuccinctID component of the artifact by taking the
933 SHA-1 hash of the identification URL as a 20-byte binary value. Note that the IdentityProviderSuccinctID
934 value, used to construct the artifact, is not encoded in hexadecimal. The AssertionHandle value is constructed
935 from a cryptographically strong random or pseudo-random number sequence (see [\[RFC1750\]](#)) generated by the
936 identity provider. The sequence consists of a value of at least eight bytes. The value should be padded to a total
937 length of 20 bytes.

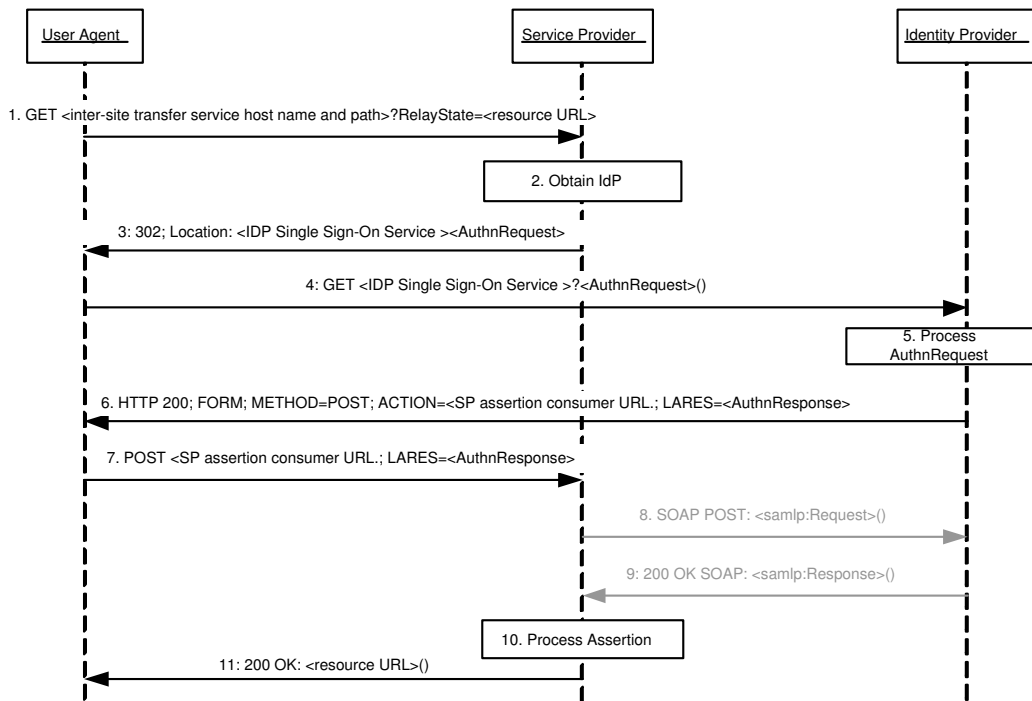
938 3.2.3. Liberty Browser POST Profile

939 The Liberty browser POST profile allows authentication information to be supplied to an identity provider without the
940 use of an artifact. Figure 3 diagrams the interactions between parties in the Liberty POST profile. This profile is an
941 adaptation of the "Browser/post profile" for SAML as documented in [SAMLBind11].

942 The following URI-based identifier MUST be used when referencing this specific profile (for example,
943 <lib:ProtocolProfile> element of the <lib:AuthnRequest> message):

944 URI: `http://projectliberty.org/profiles/brws-post`

945 The Liberty POST profile consists of a series of two interactions, the first between a user agent and an identity provider,
946 and the second directly between the user agent and the service provider.



947

948 Figure 3. Liberty browser POST profile for single sign-on

949 This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1.
950 Thus, a valid session exists for the user agent at the identity provider.

951 When implementing this profile, all processing rules defined in Section 3.2.1 for single sign-on profiles MUST be
952 followed with the exception that steps 8 and 9 MUST be omitted. Additionally, the following rules MUST be observed
953 as they relate to steps 3, 6 and 7:

954 3.2.3.1. Step 3: Single Sign-On Service with <AuthnRequest>

955 In step 3, the service provider's intersite transfer service responds and sends the user agent to the single sign-on service
956 URL at the identity provider.

957 This step may take place via an HTTP 302 redirect, a WML redirect deck or any other method that results in the user
958 agent being instructed to make an HTTP GET or POST request to the identity provider's single sign-on service.

959 This response MUST adhere to the following rules:

- 960 • The response MUST contain the identity provider's single sign-on service URL (for example, as the Location
961 header of an HTTP 302 redirect, the `action` attribute of an `HTMLform` or the `href` attribute of a `<go>` element
962 in a WML redirect deck).
- 963 • The identity provider's single sign-on service URL MUST specify `https` as the URL scheme.
- 964 **Note:**
965 Future protocols may be adopted and enabled to work within this framework. Therefore, implementers are
966 encouraged to not hardcode a reliance on `https`.
- 967 • The response MUST include one of the following:
968
- 969 • A `<query>` component containing the `<lib:AuthnRequest>` protocol message as defined in [\[LibertyProtSchema\]](#) with formatting as specified in [Section 3.1.2](#)
- 970 **Note:**
971 The `<lib:RelayState>` element of the `<lib:AuthnRequest>` message can be used by the service provider
972 to help maintain state information during the single sign-on and federation process. For example, the originally
973 requested resource (that is, `RelayState` in step 1) could be stored as the value for the `<lib:RelayState>` element,
974 which would then be returned to the service provider in the `<lib:AuthnResponse>` in step 7. The service
975 provider could then use this information to formulate the HTTP response to the user agent in step 11.
976
- 977 • An HTTP form containing the field `LAREQ` with the value of the `<lib:AuthnRequest>` protocol message
978 as defined in [\[LibertyProtSchema\]](#). The `<lib:AuthnRequest>` MUST be encoded by applying a base64
979 transformation (see [\[RFC2045\]](#)).

980 See the discussion of this step in the artifact profile for implementation examples.

981 **3.2.3.2. Step 6: Generating and Supplying the <AuthnResponse>**

982 In step 6 the identity provider generates an HTML form containing an authentication assertion that MUST be sent in
983 an HTTP 200 response to the user agent.

984 The form MUST be constructed such that it requests a POST to the service provider's assertion consumer URL
985 with form contents that contain the field `LARES` with the value being the `<lib:AuthnResponse>` protocol
986 message as defined in [\[LibertyProtSchema\]](#). The `<lib:AuthnResponse>` MUST be encoded by applying a base64
987 transformation (refer to [\[RFC2045\]](#)) to the `<lib:AuthnResponse>` and all of its elements. The service provider's
988 assertion consumer service URL used as the target of the form POST MUST specify `https` as the URL scheme; if
989 another scheme is specified, it MUST be treated as an error by the identity provider.

990 Multiple `<saml:Assertion>` elements MAY be included in the response. The identity provider MUST digitally sign
991 each of the assertions included in the response.

992 The `<saml:ConfirmationMethod>` element of the assertion MUST be set to the value specified in [\[SAMLCore11\]](#)
993 for "Assertion Bearer."

994 **3.2.3.3. Step 7: Posting the Form Containing the <AuthnResponse>**

995 In step 7 the user agent issues the HTTP POST request containing the `<lib:AuthnResponse>` to the service provider.

996 **3.2.4. Liberty-Enabled Client and Proxy Profile**

997 The Liberty-enabled client and proxy profile specifies interactions between Liberty-enabled clients and/or proxies,
998 service providers, and identity providers. See [Figure 5](#). A Liberty-enabled client is a client that has, or knows how to
999 obtain, knowledge about the identity provider that the Principal wishes to use with the service provider. In addition a

1000 Liberty-enabled client receives and sends Liberty messages in the body of HTTP requests and responses. Therefore,
1001 Liberty-enabled clients have no restrictions on the size of the Liberty protocol messages.

1002 A Liberty-enabled proxy is a HTTP proxy (typically a WAP gateway) that emulates a Liberty-enabled client. Unless
1003 stated otherwise, all statements referring to "LECP" are to be understood as statements about both Liberty-enabled
1004 clients and Liberty-enabled proxies.

1005 In some environments the successful deployment of a Liberty-Enabled proxy may require that service providers in
1006 those environments perform operations in addition to those described below. Such cases, and specific guidance for
1007 them, are covered in [\[LibertyImplGuide\]](#).

1008 The following URI-based identifier must be used when referencing this specific profile (for example,
1009 <lib:ProtocolProfile> element of the <lib:AuthnRequest> message):

1010 URI: <http://projectliberty.org/profiles/lecp>

1011 A LECP, in addition to meeting the common requirements for profiles in [Section 3.1](#), MUST indicate that it is a
1012 LECP by including a Liberty-Enabled header or entry in the value of the HTTP User-Agent header for each HTTP
1013 request it makes. The preferred method is the Liberty-Enabled header. The formats of the Liberty-Enabled header and
1014 User-Agent header entry are defined in [Section 3.2.4.1](#).

1015 **3.2.4.1. Liberty-Enabled Indications**

1016 A LECP SHOULD add the Liberty-Enabled header to each HTTP request. The Liberty-Enabled header MUST be
1017 named `Liberty-Enabled` and be defined as using Augmented BNF as specified in section 2 of [\[RFC2616\]](#).

```
1018 Liberty-Enabled = "Liberty-Enabled" ":" LIB_Version [" "," 1#Extension]  
1019 LIB_Version = "LIBV" "=" 1*absoluteURI  
1020 ; any spaces or commas in the absoluteURI MUST be escaped as defined in section 2.4 of [RFC 2396]  
1021 Extension = ExtName "=" ExtValue  
1022 ExtName = (["." host] | <any field-value but ".", ",", " or "=">) <any field-value but "=" or ",", ">  
1023 ExtValue = <any field-value but ",", ">  
1024  
1025
```

1026 The comment, field-value, and product productions are defined in [\[RFC2616\]](#). `LIB_Version` identifies the versions
1027 of the Liberty specifications that are supported by this LECP. Each version is identified by a URI. Service providers or
1028 identity providers receiving a Liberty-Enabled header MUST ignore any URIs listed in the `LIB_Version` production
1029 that they do not recognize. All LECPs compliant with this specification MUST send out, at minimum, the URI
1030 <http://projectliberty.org/specs/v1> as a value in the `LIB_Version` production. It SHOULD precede this
1031 with the URI `urn:liberty:iff:1.2` if it supports version 1.2 requests and knows that the identity providers
1032 available to it also support version 1.2 requests and responses. It MUST NOT include this URI if it knows that the
1033 identity providers available to it cannot process version 1.2 messages. The ordering of the URIs in the `LIB_Version`
1034 header is meaningful; therefore, service providers and identity providers are encouraged to use the first version in
1035 the list that they support. Supported Liberty versions are not negotiated between the LECP and the service provider.
1036 The LECP advertises what version it supports; the service provider MUST return the response for the corresponding
1037 version as defined in step 3 below.

1038 Optional extensions MAY be added to the Liberty-Enabled header to indicate new information. The value of the
1039 `ExtName` production MUST use the "host" ";" prefixed form if the new extension name has not been standardized
1040 and registered with Liberty or its designated registration authorities. The value of the host production MUST be an
1041 IP or DNS address that is owned by the issuer of the new name. By using the DNS/IP prefix, effectively namespace
1042 collisions can be prevented without the need of introducing another centralized registration agency.

1043 A LECP MAY include the Liberty-Agent header in its requests. This header provides information about the software
1044 implementing the LECP functionality and is similar to the User-Agent and Server headers in HTTP.

1045
1046 Liberty-Agent = "Liberty-Agent" ":" 1*(product | comment)
1047

1048 **Note:**

1049 The reason for introducing the new header (that is, Liberty-Enabled) rather than using User-Agent is that a
1050 LECP may be a Liberty-enabled proxy. In such a case the information about the Liberty-enabled proxy would
1051 not be in the User-Agent header. In theory the information could be in the VIA header. However, for security
1052 reasons, values in the VIA header can be collapsed, and comments (where software information would be
1053 recorded) can always be removed. As such, the VIA header is not suitable. Using the User-Agent header
1054 for a Liberty-enabled client and the Liberty-Agent header for a Liberty-enabled proxy was also discussed.
1055 However, this approach seemed too complex.

1056 Originally the Liberty-Agent header was going to be part of the Liberty-Enabled header. However, header
1057 lengths in HTTP implementations are limited; therefore, putting this information in its own header was
1058 considered the preferred approach.

1059 A LECP MAY add a Liberty-Enabled entry in the HTTP User-Agent request header. The HTTP User-Agent header is
1060 specified in [\[RFC2616\]](#). A LECP MAY include in the value of this header the Liberty-Enabled string as defined
1061 above for the Liberty-Enabled header.

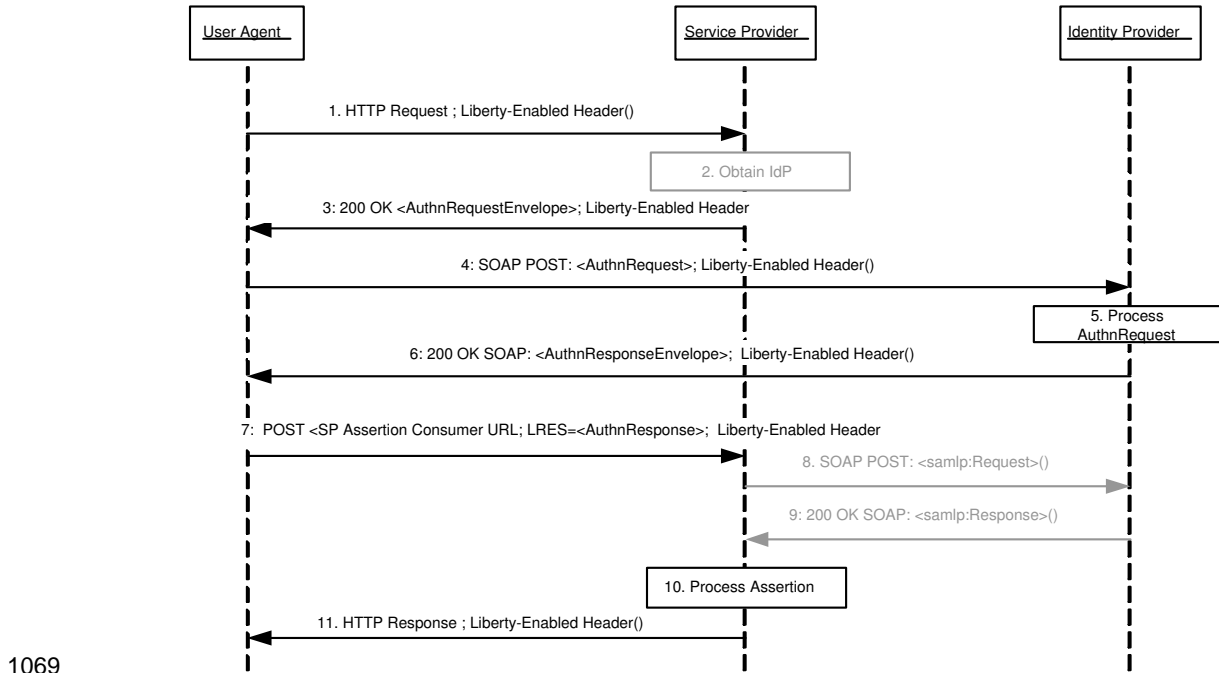
1062 **Note:**

1063 The reason for adding information to the User-Agent header is to allow for Liberty-enabled client products
1064 that must rely on a platform that cannot be instructed to insert new headers in each HTTP request.

1065 The User-Agent header is often overloaded; therefore, the Liberty-Enabled header should be the first choice
1066 for any implementation of a LECP. The entry in the User-Agent header then remains as a last resort.

1067 **3.2.4.2. Interactions**

1068 Figure 5 illustrates the Liberty-enabled client and proxy profile for single sign-on.



1069

1070

Figure 5. Liberty-enabled client and proxy profile for single sign-on

1071 This profile description assumes that the user agent has already authenticated at the identity provider prior to step 1.
1072 Thus, a valid session exists for the user agent at the identity provider.

1073 The LECP receives authentication requests from the service provider in the body of the HTTP response. The
1074 LECP submits this authentication request as a SOAP request to the identity provider. Because this SOAP re-
1075 quest is between the LECP and the identity provider, TLS authentication cannot be performed between service
1076 provider and identity provider; therefore, service providers and identity providers MUST rely on the signature of
1077 the `<lib:AuthnRequest>` and the returned `<saml:Assertion>`, respectively, for mutual authentication.

1078 When implementing this profile, processing rules for steps 5, 10, and 11 defined in [Section 3.2.1](#) for single sign-on
1079 profiles MUST be followed, while steps 2, 8, and 9 MUST be omitted. Additionally, the following rules MUST be
1080 observed as they relate to steps 1, 3, 4, 6, and 7:

1081 3.2.4.2.1. Step 1: Accessing the Service Provider

1082 In step 1, the user agent accesses the service provider with the Liberty-Enabled header (or with the Liberty-Enabled
1083 entry in the User-Agent header) included in the HTTP request.

1084 The HTTP request MUST contain only one Liberty-Enabled header. Hence if a proxy receives a HTTP request
1085 that contains a Liberty-Enabled header, it MUST NOT add another Liberty-Enabled header. However, a proxy
1086 MAY replace the Liberty-Enabled header. A proxy that replaces or adds a Liberty-Enabled header MUST process
1087 `<lib:AuthnRequest>` messages as defined in steps 3 and 4 as well as `<lib:AuthnResponse>` messages as
1088 specified in steps 6 and 7.

1089 It is RECOMMENDED that a LECP add "application/vnd.liberty-request+xml" as one of its supported
1090 content types to the Accept header.

1091 3.2.4.2.2. Step 3: HTTP Response with <AuthnRequest>

1092 In step 3, the service provider's intersite transfer service issues an HTTP 200 OK response to the user agent. The
1093 response MUST contain a single `<lib:AuthnRequestEnvelope>` with content as defined in [\[LibertyProtSchema\]](#).
1094 If a service provider receives a Liberty-Enabled header, or a User-Agent header with the Liberty-Enabled entry, the

1095 service provider MUST respond according to the Liberty-enabled client and proxy profile and include a Liberty-
1096 Enabled header in its response. Hence service providers MUST support the Liberty-enabled client and proxy profile.

1097 The processing rules and default values for the Liberty-Enabled indications are as defined in [Section 3.2.4.1](#). The
1098 service provider MAY advertise any Liberty version supported in this header, not only the version used for the specific
1099 response.

1100 The HTTP response MUST contain a Content-Type header with the value `application/vnd.liberty-request+xml`
1101 unless the LECP and service provider have negotiated a different format.

1102 A service provider MAY provide a list of identity providers it recognizes by including the `<lib:IDPList>` element
1103 in the `<lib:AuthnRequestEnvelope>`. The format and processing rules for the identity provider list MUST be as
1104 defined in [\[LibertyProtSchema\]](#).

1105 **Note:**

1106 In cases where a value for the `<lib:GetComplete>` element is provided within `<lib:IDPList>`, the URI
1107 value for this element MUST specify `https` as the URL `<scheme>`.

1108 The service provider MUST specify a URL for receiving `<AuthnResponse>` elements, locally gener-
1109 ated by the intermediary, by including the `<lib:AssertionConsumerServiceURL>` element in the
1110 `<lib:AuthnRequestEnvelope>`.

1111 The following example demonstrates the usage of the `<lib:AuthnRequestEnvelope>`:

```
1112 <?xml version="1.0" ?>
1113 <lib:AuthnRequestEnvelope xmlns:lib="urn:liberty:iff:2003-08">
1114   <lib:AuthnRequest >
1115     . . . AuthnRequest goes here . . .
1116   </lib:AuthnRequest>
1117   <lib:AssertionConsumerServiceURL>
1118     https://service-provider.com/LibertyLogin
1119   </lib:AssertionConsumerServiceURL>
1120   <lib:IDPList >
1121     . . . IdP list goes here . . .
1122   </lib:IDPList>
1123 </lib:AuthnRequestEnvelope>
```

1125 If the service provider does not support the LECP-advertised Liberty version, the service provider MUST return to the
1126 LECP an HTTP 501 response with the reason phrase "Unsupported Liberty Version."

1127 The responses in step 3 and step 6 SHOULD NOT be cached. To this end service providers and identity providers
1128 SHOULD place both "Cache-Control: no-cache" and "Pragma: no-cache" on their responses to ensure that
1129 the LECP and any intervening proxies will not cache the response.

1130 **3.2.4.2.3. Step 4: HTTP Request with `<AuthnRequest>`**

1131 In step 4, the LECP determines the appropriate identity provider to use and then issues an HTTP POST of the
1132 `<lib:AuthnRequest>` in the body of a SOAP message to the identity provider's single sign-on service URL. The
1133 request MUST contain the same `<lib:AuthnRequest>` as was received in the `<lib:AuthnRequestEnvelope>`
1134 from the service provider in step 3.

1135 **Note:**

1136 The identity provider list can be used by the LECP to create a user identifier to be presented to the Principal.
1137 For example, the LECP could compare the list of the Principal's known identities (and the identities of the
1138 identity provider that provides those identities) against the list provided by the service provider and then only
1139 display the intersection.

1140 If the LECP discovers a syntax error due to the service provider or cannot proceed any further for other reasons (for
1141 example, cannot resolve identity provider, cannot reach the identity provider, etc), the LECP MUST return to the
1142 service provider a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the desired error element as defined
1143 in [\[LibertyProtSchema\]](#). The `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the
1144 service provider's assertion consumer service URL obtained from the `<lib:AssertionConsumerServiceURL>`
1145 element of the `<lib:AuthnRequestEnvelope>`. The POST MUST be a form that contains the field LARES with
1146 the value being the `<lib:AuthnResponse>` protocol message as defined in [\[LibertyProtSchema\]](#), containing the
1147 `<samlp:Status>`. The `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to
1148 [\[RFC2045\]](#)) to the `<lib:AuthnResponse>` and all its elements.

1149 **3.2.4.2.4. Step 6: HTTP Response with `<AuthnResponse>`**

1150 In step 6, the identity provider responds to the `<lib:AuthnRequest>` by issuing an HTTP 200 OK response. The
1151 response MUST contain a single `<lib:AuthnResponseEnvelope>` in the body of a SOAP message with content as
1152 defined in [\[LibertyProtSchema\]](#).

1153 In step 6, the identity provider responds to the `<lib:AuthnRequest>` by issuing an HTTP 200 OK response. The
1154 response MUST contain a single `<lib:AuthnResponseEnvelope>` in the body of a SOAP message with content as
1155 defined in [\[LibertyProtSchema\]](#).

1156 The identity provider MUST include the Liberty-Enabled HTTP header following the same processing rules as defined
1157 in 3.2.5.1.

1158 The Content-Type MUST be set to `application/vnd.liberty-response+xml`.

1159 If the identity provider discovers a syntax error due to the service provider or LECP or cannot proceed any further
1160 for other reasons (for example, an unsupported Liberty version), the identity provider MUST return to the LECP a
1161 `<lib:AuthnResponseEnvelope>` containing a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the
1162 desired error element as defined in [\[LibertyProtSchema\]](#).

1163 **3.2.4.2.5. Step 7: Posting the Form Containing the `<AuthnResponse>`**

1164 In step 7, the LECP issues an HTTP POST of the `<lib:AuthnResponse>` that was received in the
1165 `<lib:AuthnResponseEnvelope>` SOAP response in step 6. The `<lib:AuthnResponse>` MUST
1166 be sent using a POST to the service provider's assertion consumer service URL identified by the
1167 `<lib:AssertionConsumerServiceURL>` element within the `<lib:AuthnResponseEnvelope>` *obtained*
1168 *from the identity provider in step 6*. The POST MUST be a form that contains the field LARES with the value being
1169 the `<lib:AuthnResponse>` protocol message as defined in [\[LibertyProtSchema\]](#). The `<lib:AuthnResponse>`
1170 MUST be encoded by applying a base64 transformation (refer to [\[RFC2045\]](#)) to the `<lib:AuthnResponse>` and
1171 all its elements. The service provider's assertion consumer service URL used as the target of the form POST MUST
1172 specify `https` as the URL scheme; if another scheme is specified, it MUST be treated as an error by the identity
1173 provider.

1174 If the LECP discovers an error (for example, syntax error in identity provider response), the LECP MUST return
1175 to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>` indicating the appropriate error ele-
1176 ment as defined in [\[LibertyProtSchema\]](#). The `<ProviderID>` in the `<lib:AuthnResponse>` MUST be set to
1177 `urn:liberty:iff:lecp`. The `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the
1178 service provider's assertion consumer service URL. The POST MUST be a form that contains the field named LARES
1179 with its value being the `<lib:AuthnResponse>` protocol message as defined in [\[LibertyProtSchema\]](#) with format-
1180 ting as specified 3.1.2. Any `<lib:AuthnResponse>` messages created by the identity provider MUST not be sent to
1181 the service provider.

1182 **3.3. Register Name Identifier Profiles**

1183 This section defines the profiles by which a provider may register or change a name identifier for a Principal. This
1184 message exchange is optional. During federation, the identity provider supplies an opaque handle identifying the

1185 Principle. This is the `<lib:IDPProvidedNameIdentifier>`. If neither provider involved in the federation opts
1186 to register any other name identifier, then this initial `<lib:IDPProvidedNameIdentifier>` is to be used by both
1187 providers.

1188 An identity provider may choose to register a new `<lib:IDPProvidedNameIdentifier>` at any time
1189 subsequent to federation, using this protocol. Additionally, a service provider may choose to register
1190 an `<lib:SPPProvidedNameIdentifier>`, which it expects the identity provider to use (instead of the
1191 `<lib:IDPProvidedNameIdentifier>`) when communicating with it about the Principal.

1192 Two profiles are specified: HTTP-Redirect-Based and SOAP/HTTP-based.

1193 Either the identity or service provider may initiate the register name identifier protocol. The available profiles are
1194 defined in [Section 3.3.1](#) and [Section 3.3.2](#), and vary slightly based on whether the protocol was initiated by the identity
1195 or service provider:

1196 • Register Name Identifier Initiated at Identity Provider
1197

1198 • HTTP-Redirect-Based: Relies on a HTTP 302 redirect to communicate between the identity provider and the
1199 service provider.

1200 • SOAP/HTTP-Based: Relies on a SOAP call from the identity provider to the service provider.

1201 • Register Name Identifier Initiated at Service Provider
1202

1203 • HTTP-Redirect-Based: Relies on a HTTP 302 redirect to communicate between the service provider and the
1204 identity provider.

1205 • SOAP/HTTP-Based: Relies on a SOAP call from the service provider to the identity provider.

1206 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles are essentially the
1207 same regardless of whether the profile was initiated at the service provider or at the identity provider, but the message
1208 flow directions are reversed.

1209 The register name identifier profiles make use of the following metadata elements, as defined in [\[LibertyMetadata\]](#):

1210 • `RegisterNameIdentifierProtocolProfile`: The service provider's preferred register name identifier pro-
1211 file, which should be used by the identity provider when registering a new identifier. This would specify the URI
1212 based identifier for one of the IDP Initiated register name identifier profiles.

1213 • `RegisterNameIdentifierServiceURL`: The URL used for user-agent-based Register Name Identifier Protocol
1214 profiles.

1215 • `RegisterNameIdentifierServiceReturnURL`: The provider's redirecting URL for use after HTTP name
1216 registration has taken place.

1217 • `SOAPEndpoint`: The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP
1218 messages are sent.

1219 **3.3.1. Register Name Identifier Initiated at Identity Provider**

1220 An identity provider MAY change the <lib:IDPProvidedNameIdentifier> it has assigned a Principal and
 1221 transmit that information to a service provider. The <lib:IDPProvidedNameIdentifier> MAY be changed
 1222 without changing any federations. The reasons an identity provider may wish to change the name identifier
 1223 for a Principal are implementation dependent, and thus outside the scope of this specification. Changing the
 1224 <lib:IDPProvidedNameIdentifier> MAY be accomplished in either an HTTP-Redirect-Based or SOAP/HTTP
 1225 mode.

1226 **3.3.1.1. HTTP-Redirect-Based Profile**

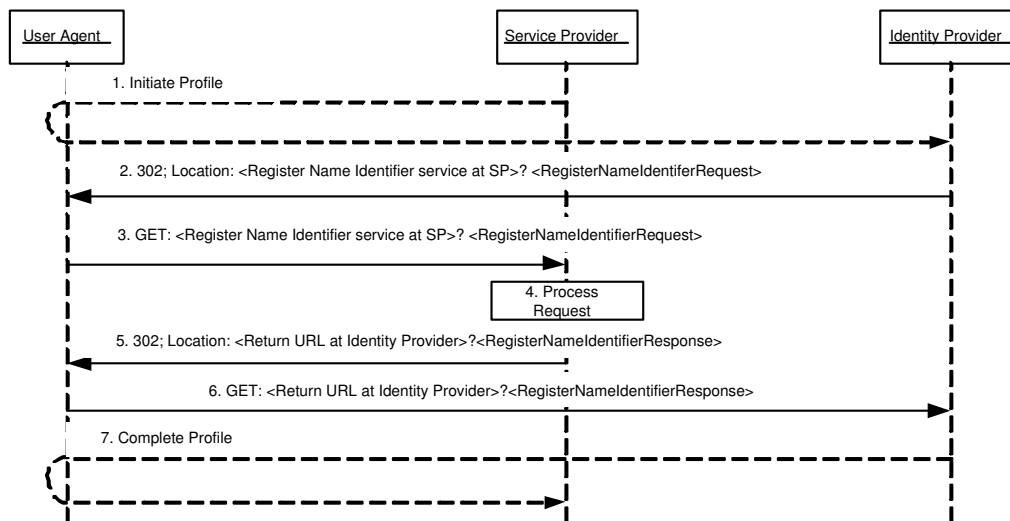
1227 A HTTP-redirect-based register name identifier profile cannot be self-initiated by an identity provider, but must be a
 1228 triggered by a message, such as an <lib:AuthnRequest>. We note that we do not normatively specify when and
 1229 how the identity provider can initiate this profile—that is left to the discretion of the identity provider. As an example,
 1230 it may be triggered by a message, such as an <lib:AuthnRequest>. When the identity provider decides to initiate
 1231 the profile in this case, it will insert this profile between the AuthnRequest/AuthnResponse transactions.

1232 The HTTP-redirect-based profile relies on using HTTP 302 redirects to communicate register name identifier messages
 1233 from the identity provider to the service provider. The HTTP-Redirect Register Name Identifier Profile (Figure 6)
 1234 illustrates this transaction.

1235 The following URI-based identifier MUST be used when referencing this specific profile:

1236 URI: `http://projectliberty.org/profiles/rni-idp-http`

1237 This URI identifier MUST be specified in the service provider metadata element RegisterNameIdentifierProtocolProfile
 1238 when the service provider intends to indicate to the identity provider a preference for receiving register name identifier
 1239 messages via a HTTP 302 redirect.



1240

1241 Figure 6. Register Name Identifier Profile.

1242 In an example scenario, the service provider makes an <lib:AuthnRequest> to the identity provider for authentication of the Principal's User Agent (step 1). The identity provider effects an <lib:IDPProvidedNameIdentifier> change in the service provider via a URL redirection. The profile is as follows:

1245 **3.3.1.1.1. Step 1: Initiate Profile**

1246 This interaction is not normatively specified as part of the profile, but shown for illustrative purposes.

1247 **3.3.1.1.2. Step 2: Redirecting to the Service Provider Register Name Identifier Service**

1248 In step 2, the identity provider redirects the user agent to the register name identifier service at the service provider.

1249 The redirection MUST adhere to the following rules:

- 1250 • The Location HTTP header MUST be set to the service provider's register name identifier service URL.
- 1251 • The service provider's register name identifier service URL MUST specify `https` as the URL scheme; if another
- 1252 scheme is specified, the identity provider MUST NOT redirect to the service provider.
- 1253 • The Location HTTP header MUST include a `<query>` component containing the `<lib:RegisterNameIdentifierRequest>`
- 1254 protocol message as defined in [\[LibertyProtSchema\]](#) with formatting as specified in [Section 3.1.2](#).

1255 The HTTP response MUST take the following form:

```
1256 <HTTP-Version> 302 <Reason Phrase>  
1257 <other headers>  
1258 Location : https://<Service Provider Register Name Identifier service URL>?<query>  
1259 <other HTTP 1.0 or 1.1 components>  
1260  
1261
```

1262 where:

1263 `<Service Provider Register Name Identifier service URL>`

1264 This element provides the host name, port number, and path components of the register name identifier service URL
1265 at the service provider.

1266 `<query>= ...<URL-encoded RegisterNameIdentifierRequest>...`

1267 The `<query>` component MUST contain a single register name identifier request.

1268 **3.3.1.1.3. Step 3: Accessing the Service Provider Register Name Identifier Service**

1269 In step 3, the user agent accesses the service provider's register name identifier service URL with the
1270 `<lib:RegisterNameIdentifierRequest>` information attached to the URL fulfilling the redirect request.

1271 **3.3.1.1.4. Step 4: Processing the Register Name Identifier Request**

1272 In step 4, the service provider MUST process the `<lib:RegisterNameIdentifierRequest>` according to the
1273 rules defined in [\[LibertyProtSchema\]](#).

1274 The service provider MAY remove the old name identifier after registering the new name identifier.

1275 **3.3.1.1.5. Step 5: Redirecting to the Identity Provider return URL with the Register Name Identifier
1276 Response**

1277 In step 5, the service provider's register name identifier service responds and redirects the user agent back to identity
1278 provider using a return URL location specified in the `RegisterNameIdentifierServiceReturnURL` metadata element. If
1279 the URL-encoded `<lib: RegisterNameIdentifierRequest>` message received in step 3 contains a parameter
1280 named `RelayState`, then the service provider MUST include a `<query>` component containing the same `RelayState`
1281 parameter and its value in its response to the identity provider.

1282 The redirection MUST adhere to the following rules:

- 1283 • The Location HTTP header MUST be set to the identity providers return URL specified in the `RegisterNameIden-`
- 1284 `tifierServiceReturnURL` metadata element.

1285 • The identity provider's return URL MUST specify `https` as the URL scheme; if another scheme is specified, the
1286 service provider MUST NOT redirect to the identity provider.

1287 • The Location HTTP header MUST include a `<query>` component containing the `<lib:RegisterNameIdentifierResponse>`
1288 protocol message as defined in [LibertyProtSchema] with formatting as specified in Section 3.1.2.

1289 The HTTP response MUST take the following form:

```
1290  
1291 <HTTP-Version> 302 <Reason Phrase>  
1292 <other headers>  
1293 Location : https://<Identity Provider Service Return URL >?<query>  
1294 <other HTTP 1.0 or 1.1 components>  
1295
```

1296 where:

```
1297 <Identity Provider Service Return URL>
```

1298 This element provides the host name, port number, and path components of the return URL at the identity provider.

```
1299 <query>= ...<URL-encoded RegisterNameIdentifierResponse>...
```

1300 The `<query>` component MUST contain a single register name identifier response. The `<URL-encoded`
1301 `RegisterNameIdentifierResponse>` component MUST contain the identical `RelayState` parameter and its value
1302 that was received in the URL-encoded register name identifier message obtained in step 3. If no `RelayState` parameter
1303 was provided in the step 3 message, then a `RelayState` parameter MUST NOT be specified in the `<URL-encoded`
1304 `RegisterNameIdentifierResponse>`.

1305 **3.3.1.1.6. Step 6: Accessing the Identity Provider return URL with the Register Name Identifier** 1306 **Response**

1307 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect request.

1308 **3.3.1.1.7. Step 7: Complete profile**

1309 This concludes the initial sequence, which triggered the initiation of this profile.

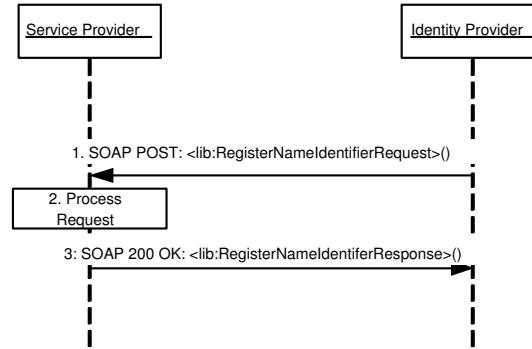
1310 **3.3.1.2. SOAP/HTTP-Based Profile**

1311 The following URI-based identifier MUST be used when referencing this specific profile:

```
1312 URI: http://projectliberty.org/profiles/rni-idp-soap
```

1313 This URI identifier MUST be specified in the service provider metadata element `RegisterNameIdentifierProtocolPro-`
1314 `file` when the service provider intends to indicate to the identity provider a preference for receiving register name
1315 identifier messages via SOAP over HTTP.

1316 The steps involved in the SOAP/HTTP-based profile MUST utilize the SOAP binding for Liberty as defined in
1317 Section 2.1. See Figure 7.



1318

1319

Figure 7. SOAP/HTTP-based profile for registering name identifiers

1320 3.3.1.2.1. Step 1 Initiate Profile

1321 In step 1, the identity provider sends a `<lib:RegisterNameIdentifierRequest>` protocol mes-
1322 sage to the service provider's SOAP endpoint specifying `<lib:SPProvidedNameIdentifier>` ,
1323 `<lib:IDPProvidedNameIdentifier>`, and `<lib:OldProvidedNameIdentifier>` as defined in [[LibertyProtSchema](#)]. The `<lib:SPProvidedNameIdentifier>` will only contain a value if the service provider has
1325 previously used the register name identifier profile.

1326 3.3.1.2.2. Step 2: Process Request

1327 Service provider records new `<lib:IDPProvidedNameIdentifier>`.

1328 3.3.1.2.3. Step 3: Response to Register Name Identifier

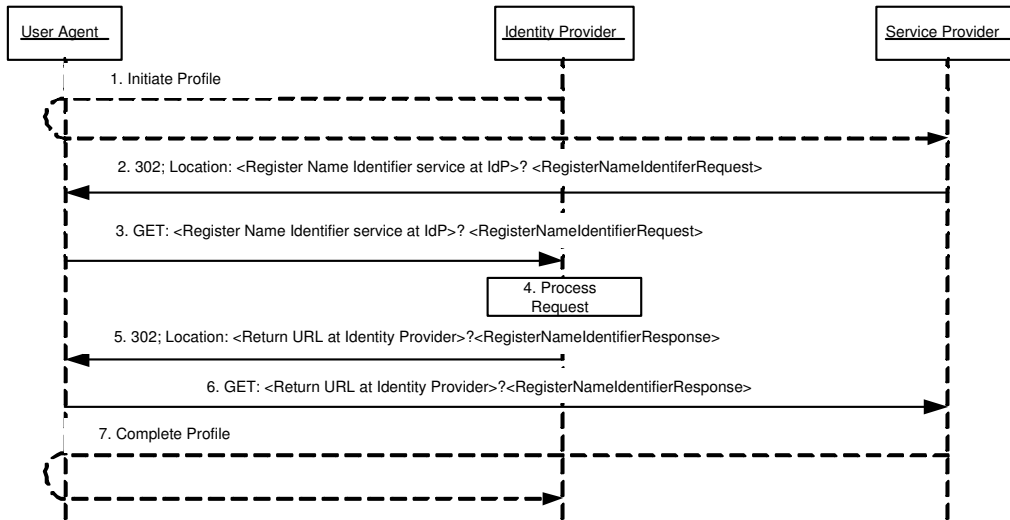
1329 The service provider, after successfully registering the new `<lib:IDPProvidedNameIdentifier>` provided by the
1330 identity provider, MUST respond with a `<lib:RegisterNameIdentifierResponse>` according to the processing
1331 rules defined in [[LibertyProtSchema](#)].

1332 3.3.2. Register Name Identifier Initiated at ServiceProvider

1333 A service provider may register, or change a `<lib:SPProvidedNameIdentifier>` which is a name iden-
1334 tifier it expects the identity provider to use when communicating with it about the Principal. Until it
1335 registers a `<lib:SPProvidedNameIdentifier>`, an identity provider will continue to use the current
1336 `<lib:IDPProvidedNameIdentifier>` when referring to the Principal.

1337 3.3.2.1. HTTP-Redirect-Based Profile

1338 The HTTP-redirect-based profile relies on the use of a HTTP 302 redirect to communicate a register name identifier
1339 message from the service provider to the identity provider.



1340

1341

Figure 8. SP-Initiated Register Name Identifier Profile.

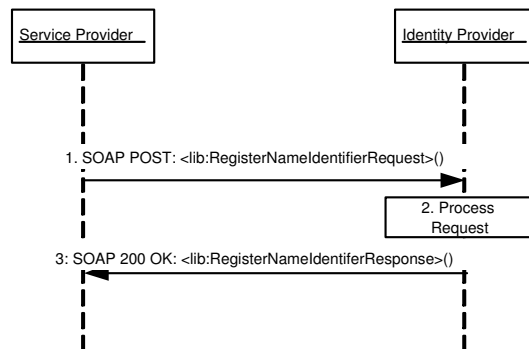
1342 The following URI-based identifier MUST be used when referencing this specific profile:

1343 URI: `http://projectliberty.org/profiles/rni-sp-http`

1344 A HTTP-redirect-based register name identifier profile can be self-initiated by a service provider to change the
 1345 `<lib:SPProvidedNameIdentifier>`. This does not normatively specify when and how the service provider
 1346 can initiate this profile; that is left to the discretion of the service provider. The HTTP-redirect-based profile relies on
 1347 using HTTP 302 redirects to communicate register name identifier messages from the service provider to the identity
 1348 provider. The service provider effects a `<lib:SPProvidedNameIdentifier>` change in the identity provider
 1349 via a URL redirection. For a discussion of the interactions and processing steps, refer to [Section 3.3.1.1](#). When
 1350 reviewing that profile, interchange all references to service provider and identity provider in the interaction diagram
 1351 and processing steps 2-6. See Figure 8. Note that in step 4 the old `SPProvidedNameIdentifier` SHOULD be removed
 1352 at the IdP.

1353 3.3.2.2. SOAP/HTTP-Based Profile

1354 The SOAP/HTTP-based profile relies on using SOAP over HTTP to communicate register name identifier messages
 1355 from the service provider to the identity provider. For a discussion of the interactions and processing steps, refer to
 1356 [Section 3.3.1.2](#). When reviewing that profile, interchange all references to service provider and identity provider in
 1357 the interaction diagram and processing steps. See [Figure 9](#).



1358

1359

Figure 9. SP-Initiated SOAP/HTTP-based profile for registering name identifiers

1360 The following URI-based identifier MUST be used when referencing this specific profile:

1361 URI: <http://projectliberty.org/profiles/rni-sp-soap>

1362 In step 1, the service provider sends a `<lib:RegisterNameIdentifierRequest>` protocol mes-
1363 sage to the identity provider's SOAP endpoint specifying `<lib:SPProvidedNameIdentifier>` ,
1364 `<lib:IDPProvidedNameIdentifier>`, and `<lib:OldProvidedNameIdentifier>` as defined in [[LibertyProtSchema](#)]. The `<lib:OldProvidedNameIdentifier>` will only contain a value if the service provider has
1366 previously used the register name identifier profile.

1367 3.4. Identity Federation Termination Notification Profiles

1368 The Liberty identity federation termination notification profiles specify how service providers and identity providers
1369 are notified of federation termination (also known as defederation).

1370 **Note:**

1371 Other means of federation termination are possible, such as federation expiration and termination of business
1372 agreements between service providers and identity providers. These means of federation termination are
1373 outside the scope of this specification.

1374 Identity federation termination can be initiated at either the identity provider or the service provider. The Principal
1375 SHOULD have been authenticated by the provider at which identity federation termination is being initiated. The
1376 available profiles are defined in [Section 3.4.1](#) and [Section 3.4.2](#), depending on whether the identity federation
1377 termination notification process was initiated at the identity provider or service provider:

1378 • Federation Termination Notification Initiated at Identity Provider
1379

1380 • HTTP-Redirect-Based: Relies on a HTTP 302 redirect to communicate between the identity provider and the
1381 service provider.

1382 • SOAP/HTTP-Based: Relies on a SOAP call from the identity provider to the service provider.

1383 • Federation Termination Notification Initiated at Service Provider
1384

1385 • HTTP-Redirect-Based: Relies on a HTTP 302 redirect to communicate between the service provider and the
1386 identity provider.

1387 • SOAP/HTTP-Based: Relies on a SOAP call from the service provider to the identity provider.

1388 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles are essentially the
1389 same regardless of whether federation termination notification was initiated at the service provider or at the identity
1390 provider.

1391 The identity federation termination notification profiles make use of the following metadata elements, as defined in
1392 [[LibertyProtSchema](#)]:

1393 • `FederationTerminationServiceURL` - The URL at the service provider or identity provider to which identity
1394 federation termination notifications are sent. It is documented in these profiles as "federation termination service
1395 URL."

- 1396 • FederationTerminationServiceReturnURL - The URL used by the service provider or identity provider
- 1397 when redirecting the user agent at the end of the federation termination notification profile process.

- 1398 • FederationTerminationNotificationProtocolProfile - Used by the identity provider to determine
- 1399 which federation termination notification profile MUST be used when communicating with the service provider.

- 1400 • SOAPEndpoint - The SOAP endpoint location at the service provider or identity provider to which Liberty SOAP
- 1401 messages are sent.

1402 **3.4.1. Federation Termination Notification Initiated at Identity Provider**

1403 The profiles in Section 3.4.1.1 and Section 3.4.1.2 are specific to identity federation termination when initiated at the
 1404 identity provider. Effectively, when using these profiles, the identity provider is stating to the service provider that it
 1405 will no longer provide the Principal's identity information to the service provider and that the identity provider will no
 1406 longer respond to any requests by the service provider on behalf of the Principal.

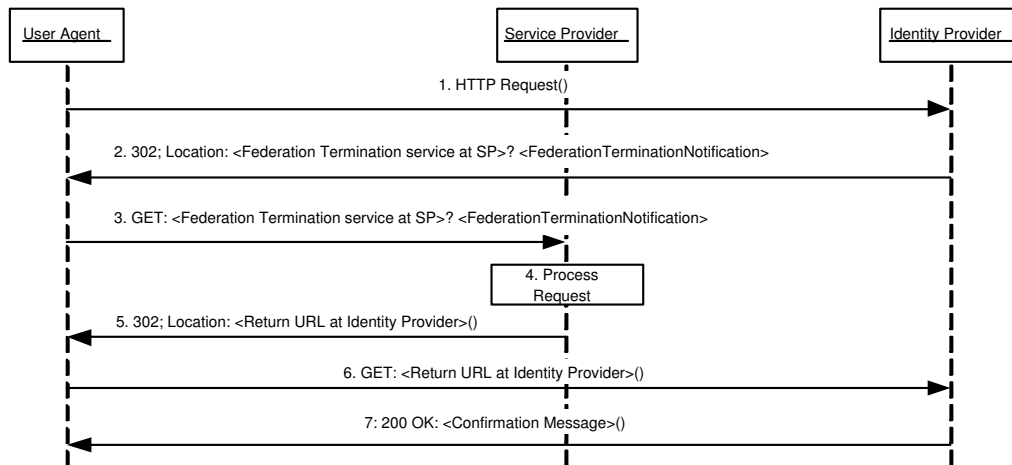
1407 **3.4.1.1. HTTP-Redirect-Based Profile**

1408 The HTTP-redirect-based profile relies on using HTTP 302 redirect to communicate federation termination notification
 1409 messages from the identity provider to the service provider. See Figure 10.

1410 The following URI-based identifier MUST be used when referencing this specific profile:

1411 URI: `http://projectliberty.org/profiles/fedterm-idp-http`

1412 This URI identifier MUST be specified in the service provider metadata element FederationTerminationNotification-
 1413 ProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving feder-
 1414 ation termination notifications via a HTTP 302 redirect.



1415

1416 Figure 10. HTTP-redirect-based profile for federation termination

1417 This profile description assumes the following preconditions:

- 1418 • The Principal's identity at the service provider is federated with his/her identity at the identity provider.
- 1419 • The Principal has requested to the identity provider that the federation be terminated.
- 1420 • The Principal has authenticated with the identity provider.

1421 **3.4.1.1.1. Step 1: Accessing the Federation Termination Service**

1422 In step 1, the user agent accesses the identity federation termination service URL at the identity provider specifying
1423 the service provider with which identity federation termination should occur. How the service provider is specified is
1424 implementation-dependent and, as such, is out of the scope of this specification.

1425 **3.4.1.1.2. Step 2: Redirecting to the Service Provider**

1426 In step 2, the identity provider's federation termination service URL responds and redirects the user agent to the
1427 federation termination service at the service provider.

1428 The redirection MUST adhere to the following rules:

- 1429 • The Location HTTP header MUST be set to the service provider's federation termination service URL.
- 1430 • The service provider's federation termination service URL MUST specify https as the URL scheme; if another
1431 scheme is specified, the identity provider MUST NOT redirect to the service provider.
- 1432 • The Location HTTP header MUST include a <query> component containing the
1433 <lib:FederationTerminationNotification> protocol message as defined in [LibertyProtSchema] with
1434 formatting as specified in Section 3.1.2.

1435 The HTTP response MUST take the following form:

```
1436  
1437 <HTTP-Version> 302 <Reason Phrase>  
1438 <other headers>  
1439 Location : https://<Service Provider Federation Termination service URL>?<query>  
1440 <other HTTP 1.0 or 1.1 components>  
1441
```

1442 where:

1443 <Service Provider Federation Termination service URL>

1444 This element provides the host name, port number, and path components of the federation termination service URL at
1445 the service provider.

1446 <query>= ...<URL-encoded FederationTerminationNotification>...

1447 The <query> component MUST contain a single terminate federation request.

1448 **3.4.1.1.3. Step 3: Accessing the Service Provider Federation Termination Service**

1449 In step 3, the user agent accesses the service provider's federation termination service URL with the
1450 <lib:FederationTerminationNotification> information attached to the URL fulfilling the redirect re-
1451 quest.

1452 **3.4.1.1.4. Step 4: Processing the Notification**

1453 In step 4, the service provider MUST process the <lib:FederationTerminationNotification> according to
1454 the rules defined in [LibertyProtSchema].

1455 The service provider MAY remove any locally stored references to the name identifier it received from the identity
1456 provider in the <lib:FederationTerminationNotification>.

1457 **3.4.1.1.5. Step 5: Redirecting to the Identity Provider Return URL**

1458 In step 5, the service provider's federation termination service responds and redirects the user agent back to identity
1459 provider using a return URL location specified in the FederationTerminationServiceReturnURL metadata element.
1460 If the URL-encoded `<lib:FederationTerminationNotification>` message received in step 3 contains a
1461 parameter named RelayState, then the service provider MUST include a `<query>` component containing the same
1462 RelayState parameter and its value in its response to the identity provider.

1463 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this redirect is to return
1464 the user agent to the identity provider where the federation termination process began.

1465 The HTTP response MUST take the following form:

```
1466  
1467 <HTTP-Version> 302 <Reason Phrase>  
1468 <other headers>  
1469 Location : https://<Identity Provider Service Return URL ?><query>  
1470 <other HTTP 1.0 or 1.1 components>  
1471  
1472
```

1473 where:

1474 `<Identity Provider Service Return URL>`

1475 This element provides the components of the return URL at the identity provider.

1476 `<query>= . . .RelayState=<. . .>`

1477 The `<query>` component MUST contain the identical RelayState parameter and its value that was received in the
1478 URL-encoded federation termination message obtained in step 3. If no RelayState parameter was provided in the step
1479 3 message, then a RelayState parameter MUST NOT be specified in the `<query>` component.

1480 **3.4.1.1.6. Step 6: Accessing the Identity Provider Return URL**

1481 In step 6, the user agent accesses the identity provider's return URL location fulfilling the redirect request.

1482 **3.4.1.1.7. Step 7: Confirmation**

1483 In step 7, the user agent is sent an HTTP response that confirms the requested action of identity federation termination
1484 with the specific service provider.

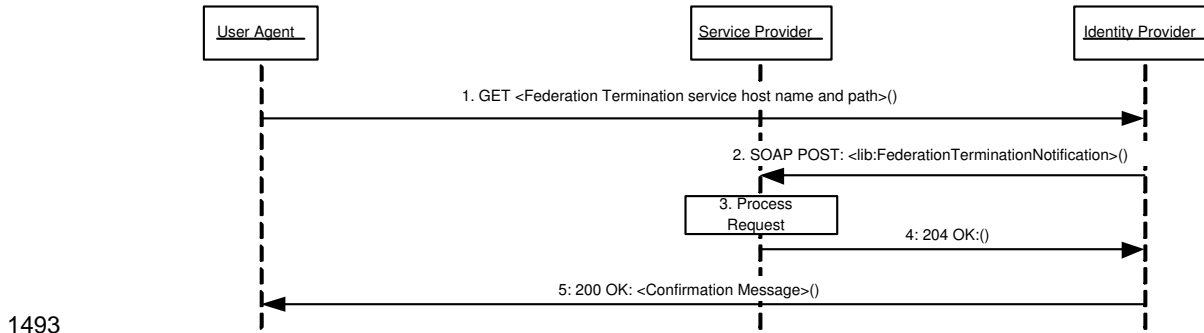
1485 **3.4.1.2. SOAP/HTTP-Based Profile**

1486 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate federation termination
1487 notification messages from the identity provider to the service provider. See Figure 11.

1488 The following URI-based identifier MUST be used when referencing this specific profile:

1489 URI: `http://projectliberty.org/profiles/fedterm-idp-soap`

1490 This URI identifier MUST be specified in the service provider metadata element FederationTerminationNotification-
1491 ProtocolProfile when the service provider intends to indicate to the identity provider a preference for receiving feder-
1492 ation termination notifications via SOAP over HTTP.



1493

1494

Figure 11. SOAP/HTTP-based profile for federation termination

1495 This profile description assumes the following preconditions:

- 1496 • The Principal’s identity at the service provider is federated with his/her identity at the identity provider.
- 1497 • The Principal’s identity at the service provider is federated with his/her identity at the identity provider.
- 1498 • The Principal has authenticated with the identity provider.

1499 **3.4.1.2.1. Step 1: Accessing the Federation Termination Service**

1500 In step 1, the user agent accesses the identity federation termination service URL at the identity provider specifying
 1501 the service provider for with which identity federation termination should occur. How the service provider is specified
 1502 is implementation-dependent and, as such, is out of the scope of this specification.

1503 **3.4.1.2.2. Step 2: Notification of Federation Termination**

1504 In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message to the service provider’s
 1505 SOAP endpoint. The SOAP message MUST contain exactly one `<lib:FederationTerminationNotification>`
 1506 element in the SOAP body and adhere to the construction rules defined in [\[LibertyProtSchema\]](#).

1507 If a SOAP fault occurs, the identity provider SHOULD employ best effort to resolve the fault condition and resend the
 1508 federation termination notification message to the service provider.

1509 **3.4.1.2.3. Step 3: Processing the Notification**

1510 In step 3, the service provider MUST process the `<lib:FederationTerminationNotification>` according to
 1511 the rules defined in [\[LibertyProtSchema\]](#).

1512 The service provider MAY remove any locally stored references to the name identifier it received from the identity
 1513 provider in the `<lib:FederationTerminationNotification>`.

1514 **3.4.1.2.4. Step 4: Responding to the Notification**

1515 In step 4, the service provider MUST respond to the `<lib:FederationTerminationNotification>` with a HTTP
 1516 204 OK response.

1517 **3.4.1.2.5. Step 5: Confirmation**

1518 In step 5, the user agent is sent an HTTP response that confirms the requested action of identity federation termination
 1519 with the specific service provider.

1520 **3.4.2. Federation Termination Notification Initiated at Service Provider**

1521 The profiles in 3.4.2.1 and 3.4.2.2 are specific to identity federation termination notification when initiated by a
1522 Principal at the service provider. Effectively, when using this profile, the service provider is stating to the identity
1523 provider that the Principal has requested that the identity provider no longer provide the Principal's identity information
1524 to the service provider and that service provider will no longer ask the identity provider to do anything on the behalf
1525 of the Principal.

1526 It is RECOMMENDED that the service provider, after initiating or receiving a federation termination notification,
1527 invalidate the local session for the Principal that was authenticated at the identity provider with which federation has
1528 been terminated. If the Principal was locally authenticated at the service provider, the service provider MAY continue
1529 to maintain a local session for the Principal. If the Principal wants to engage in a single sign-on session with identity
1530 provider again, the service provider MUST first federate with identity provider the given Principal.

1531 **3.4.2.1. HTTP-Redirect-Based Profile**

1532 The HTTP-redirect-based profile relies on the use of a HTTP 302 redirect to communicate a federation termination
1533 notification message from the service provider to the identity provider. For a discussion of the interactions and
1534 processing steps, refer to [Section 3.4.1.1](#). When reviewing that profile, interchange all references to service provider
1535 and identity provider in the interaction diagram and processing steps.

1536 The following URI-based identifier MUST be used when referencing this specific profile:

1537 URI: `http://projectliberty.org/profiles/fedterm-sp-http`

1538 This URI identifier is really only meant for service provider consumption and as such is not needed in any provider
1539 metadata.

1540 **3.4.2.2. SOAP/HTTP-Based Profile**

1541 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate federation termination
1542 notification messages from the service provider to the identity provider. For a discussion of the interactions and
1543 processing steps, refer to 3.4.1.2. When reviewing that profile, interchange all references to service provider and
1544 identity provider in the interaction diagram and processing steps.

1545 The following URI-based identifier MUST be used when referencing this specific profile:

1546 URI: `http://projectliberty.org/profiles/fedterm-sp-soap`

1547 This URI identifier is really only meant for service provider consumption and as such is not needed in any provider
1548 metadata.

1549 **3.5. Single Logout Profiles**

1550 The single logout profiles synchronize session logout functionality across all sessions that were authenticated by a
1551 particular identity provider. The single logout can be initiated at either the identity provider or the service provider.
1552 In either case, the identity provider will then communicate a logout request to each service provider with which it
1553 has established a session for the Principal. The negotiation of which single logout profile the identity provider uses
1554 to communicate with each service provider is based upon the SingleLogoutProtocolProfile provider metadata element
1555 defined in [[LibertyProtSchema](#)].

1556 The available profiles are defined in [Section 3.5.1](#) and [Section 3.5.2](#), depending on whether the single logout is initiated
1557 at the identity provider or service provider:

- 1558 • *Single Logout Initiated at Identity Provider*

1559

- 1560 • HTTP-Based: Relies on using either HTTP 302 redirects or HTTP GET requests to communicate logout
1561 requests from an identity provider to the service providers.
- 1562 • SOAP/HTTP-Based: Relies on SOAP over HTTP messaging to communicate logout requests from an identity
1563 provider to the service providers.
- 1564 • *Single Logout Initiated at Service Provider*
1565
- 1566 • HTTP-Redirect-Based: Relies on a HTTP 302 redirect to communicate a logout request with the identity provider.
- 1567 • SOAP/HTTP-Based: Relies on SOAP over HTTP messaging to communicate a logout request from a service
1568 provider to an identity provider.
- 1569 • `SingleLogoutServiceURL` — The URL at the service provider or identity provider to which single logout
1570 requests are sent. It is described in these profiles as "single logout service URL."
- 1571 • `SingleLogoutServiceReturnURL`— The URL used by the service provider when redirecting the user agent to
1572 the identity provider at the end of the single logout profile process.
- 1573 • `SingleLogoutProtocolProfile` — Used by the identity provider to determine which single logout request
1574 profile **MUST** be used when communicating with the service provider.
- 1575 • `SOAPEndpoint` — The SOAP endpoint location at the service provider or identity provider to which Liberty
1576 SOAP messages are sent.

1577 **3.5.1. Single Logout Initiated at Identity Provider**

1578 The profiles in 3.5.1.1 through 3.5.1.2 are specific to a single logout when initiated by a user agent at the identity
1579 provider.

1580 **3.5.1.1. HTTP-Based Profile**

1581 The HTTP-based profile defines two possible implementations that an identity provider may use. The first
1582 implementation relies on using HTTP 302 redirects, while the second uses HTTP GET requests. The choice of
1583 implementation is entirely dependent upon the type of user experience the identity provider provides.

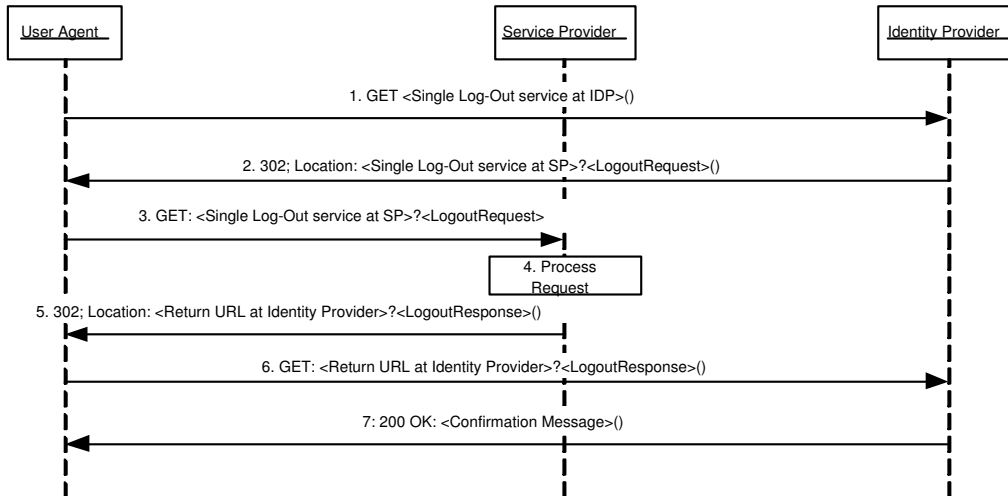
1584 The following URI-based identifier **MUST** be used when referencing either implementation for this specific profile:

1585 URI: `http://projectliberty.org/profiles/slo-idp-http`

1586 This URI identifier **MUST** be specified in the service provider metadata element `SingleLogOutProtocolProfile` when
1587 the service provider intends to indicate to the identity provider a preference for receiving logout requests via either a
1588 HTTP redirect or a HTTP GET.

1589 **3.5.1.1.1. HTTP-Redirect Implementation**

1590 The HTTP-Redirect implementation uses HTTP 302 redirects to communicate a logout request to each service provider
1591 for which the identity provider has provided authentication assertions during the Principal's current session if the
1592 service provider indicated a preference to receive logout requests via the HTTP based profile. See Figure 12.



1593

1594

Figure 12. HTTP-Redirect implementation for single logout initiated at identity provider

1595 **Notes:**

1596 Steps 2 through 6 may be an iterative process for requesting logouts by each service provider that has been
 1597 issued authentication assertions during the Principal’s current session and has indicated a preference to receive
 1598 logout requests via the HTTP based profile.

1599 [RFC2616] indicates a client should detect infinite redirection loops because such loops generate network
 1600 traffic for each redirection. This requirement was introduced because previous versions of the specification
 1601 recommended a maximum of five redirections. Content developers should be aware that some clients might
 1602 implement such a fixed limitation.

1603 **3.5.1.1.1. Step 1: Accessing the Single Logout Service at the Identity Provider**

1604 In step 1, the user agent accesses the single logout service URL at the identity provider indicating that all service
 1605 providers for which this identity provider has provided authentication assertions during the Principal’s current session
 1606 must be notified of session termination.

1607 **3.5.1.1.2. Step 2: Redirecting to the Single Logout Service at the Service Provider**

1608 In step 2, the identity provider’s single logout service responds and redirects the user agent to the single logout service
 1609 URL at each service provider for which the identity provider has provided an authentication assertion during the
 1610 Principal’s current session with the identity provider.

1611 The redirections MUST adhere to the following rules:

- 1612 • The Location HTTP header MUST be set to the service provider’s single logout service URL.
- 1613 • The service provider’s single logout service URL MUST specify https as the URL scheme; if another scheme is
 1614 specified, the identity provider MUST NOT redirect to the service provider.
- 1615 • The Location HTTP header MUST include a <query> component containing the <lib:LogoutRequest>
 1616 protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

1617 The HTTP response MUST take the following form:

```
1618  
1619 <HTTP-Version> 302 <Reason Phrase>  
1620 <other headers>  
1621 Location : https://<Service Provider Single Log-Out service URL>?<query>  
1622 <other HTTP 1.0 or 1.1 components>  
1623
```

1624 where:

1625 <Service Provider Single Log-Out service URL>

1626 This element provides the host name, port number, and path components of the single logout service URL at the
1627 service provider.

1628 <query>= ...<URL-encoded LogoutRequest>...

1629 The < query> MUST contain a single logout request.

1630 **3.5.1.1.1.3. Step 3: Accessing the Service Provider Single Logout Service**

1631 In step 3, the user agent accesses the service provider's single logout service URL with the <lib:LogoutRequest>
1632 information attached to the URL fulfilling the redirect request.

1633 **3.5.1.1.1.4. Step 4: Processing the Request**

1634 In step 4, the service provider MUST process the <lib:LogoutRequest> according to the rules defined in
1635 [\[LibertyProtSchema\]](#).

1636 The service provider MUST invalidate the session(s) of the Principal referred to in the name identifier it received from
1637 the identity provider in the <lib:LogoutRequest>.

1638 **3.5.1.1.1.5. Step 5: Redirecting to the Identity Provider Return URL**

1639 In step 5, the service provider's single logout service responds and redirects the user agent back to the identity provider
1640 using the return URL location obtained from the SingleLogoutServiceReturnURL metadata element. If the URL-
1641 encoded <lib:LogoutRequest> message received in step 3 contains a parameter named RelayState, then the service
1642 provider MUST include a <query> component containing the same RelayState parameter and its value in its response
1643 to the identity provider.

1644 The purpose of this redirect is to return the user agent to the identity provider so that the single logout process may
1645 continue in the same fashion with other service providers.

1646 The HTTP response MUST take the following form:

```
1647  
1648 <HTTP-Version> 302 <Reason Phrase>  
1649 <other headers>  
1650 Location : https://<Identity Provider Service Return URL>?<query>  
1651 <other HTTP 1.0 or 1.1 components>
```

1652 where:

1653 <Identity Provider Service Return URL>

1654 This element provides the host name, port number, and path components of the return URL at the identity provider.

1655 <query>= ...<URL-encoded LogoutResponse>

1656 The <query> component MUST contain a single logout response. The <URL-encoded LogoutResponse> MUST
 1657 contain the identical RelayState parameter and its value that was received in the URL-encoded logout request message
 1658 obtained in step 3. If no RelayState parameter was provided in the step 3 message, then a RelayState parameter MUST
 1659 NOT be specified in the <URL-encoded LogoutResponse>.

1660 **3.5.1.1.1.6. Step 6: Accessing the Identity Provider Return URL**

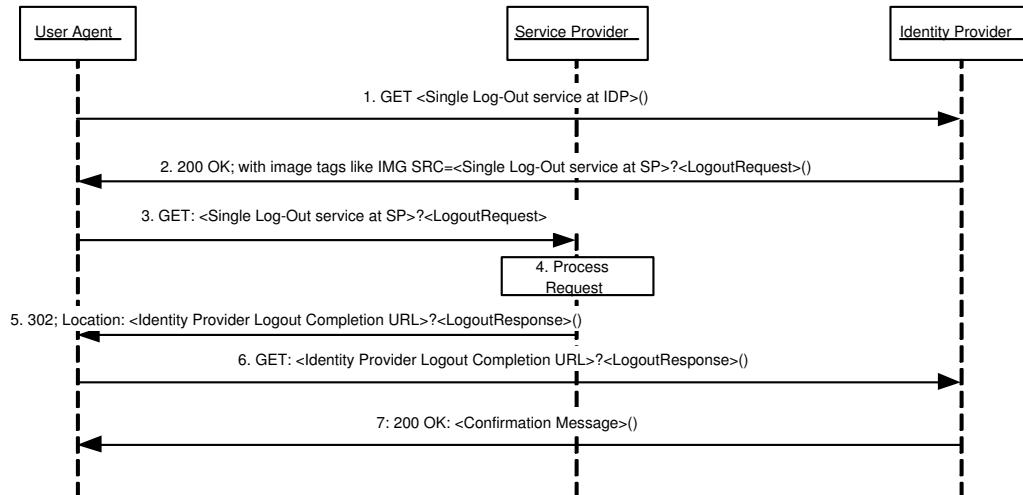
1661 In step 6, the user agent accesses the identity provider’s return URL location fulfilling the redirect request.

1662 **3.5.1.1.1.7. Step 7: Confirmation**

1663 In step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been
 1664 completed.

1665 **3.5.1.1.2. HTTP-GET Implementation**

1666 The HTTP-GET implementation uses HTTP GET requests to communicate logout requests to each service provider
 1667 for which the identity provider has provided authentication during the Principal’s current session if the service provider
 1668 indicated a preference to receive logout requests via the HTTP based profile. See Figure 13.



1669

1670 Figure 13. HTTP-GET implementation for single logout initiated at identity provider

1671 **Note:**

1672 Steps 3 through 7 may be an iterative process for requesting logout of each service provider that has been
 1673 issued authentication assertions during the Principal’s current session and has indicated a preference to receive
 1674 logout requests via the HTTP based profile.

1675 **3.5.1.1.2.1. Step 1: Accessing the Single Logout Service at the Identity Provider**

1676 In step 1, the user agent accesses the single logout service URL at the identity provider indicating that all service
 1677 providers for which this identity provider has provided authentication assertions during the Principal’s current session
 1678 must be notified of session termination and requested to logout the Principal.

1679 **3.5.1.1.2.2. Step 2: HTML Page Returned to User Agent with Image Tags**

1680 In step 2, the identity provider’s single logout service responds with an HTML page that includes image tags
 1681 referencing the logout service URL for each of the service providers for which the identity provider has provided
 1682 an authentication assertion during the Principal’s current session. The list of image tags MUST be sent in a standard
 1683 HTTP 200 response to the user agent.

1684 The image tag loads on the HTML page MUST adhere to the following rules:

- 1685 • The SRC attribute MUST be set to the specific service provider's single logout service URL.
- 1686 • The service provider's single logout service URL MUST specify https as the URL scheme.
- 1687 • The service provider's single logout service URL MUST include a `<query>` component containing the
- 1688 `<lib:LogoutRequest>` protocol message as defined in [\[LibertyProtSchema\]](#) with formatting as specified in
- 1689 3.1.2.

1690 **3.5.1.1.2.3. Step 3: Accessing the Service Provider Single Logout Service**

1691 In step 3, the user agent, as a result of each image load, accesses the service provider's single logout service URL with
1692 `<lib:LogoutRequest>` information attached to the URL. This step may occur multiple times if the HTTP response
1693 includes multiple image tag statements (one for each service provider that has been issued authentication assertions
1694 during the Principal's current session).

1695 **3.5.1.1.2.4. Step 4: Processing the Request**

1696 In step 4, the service provider MUST process the `<lib:LogoutRequest>` according to the rules defined in
1697 [\[LibertyProtSchema\]](#).

1698 The service provider MUST invalidate the session of the Principal referred to in the name identifier it received from
1699 the identity provider in the `<lib:LogoutRequest>`.

1700 **3.5.1.1.2.5. Step 5: Redirecting to the Identity Provider Logout Completion URL**

1701 In step 5, the service provider's single logout service responds and redirects the image load back to the identity
1702 provider's logout completion URL. This location will typically point to an image that will be loaded by the user agent
1703 to indicate that the logout is complete (for example, a checkmark).

1704 The logout completion URL is obtained from the `SingleLogoutServiceReturnURL` metadata element.

1705 The HTTP response MUST take the following form:

```
1706 <HTTP-Version> 302 <Reason Phrase>  
1707 <other headers>  
1708 Location : https://<Identity Provider Logout Completion URL>?<query>  
1709 <other HTTP 1.0 or 1.1 components>
```

1711 where:

1712 `<Identity Provider Logout Completion URL>`

1713 This element provides the host name, port number, and path components of the identity provider logout completion
1714 URL at the identity provider.

1715 `<query>=...<URL-encoded LogoutResponse>`

1716 The `<query>` component MUST contain a single logout response. The `<URL-encoded LogoutResponse>`
1717 component MUST contain the identical `RelayState` parameter and its value that was received in the URL-encoded
1718 logout request message obtained in step 3. If no `RelayState` parameter was provided in step 3 then a `RelayState`
1719 message MUST NOT be specified in the `<URL-encoded LogoutResponse>`.

1720 **3.5.1.1.2.6. Step 6: Accessing the Identity Provider Logout Completion URL**

1721 In step 6, the user agent accesses the identity provider's logout completion URL fulfilling the redirect request.

1722 3.5.1.1.2.7. Step 7: Confirmation

1723 In step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been
1724 completed.

1725 Note:

1726 One method for seamlessly returning the user agent back to the identity provider is for the HTML page
1727 generated in step 2 to include a script that runs when the page is completely loaded (all logouts completed)
1728 that will initiate the redirect to the identity provider.

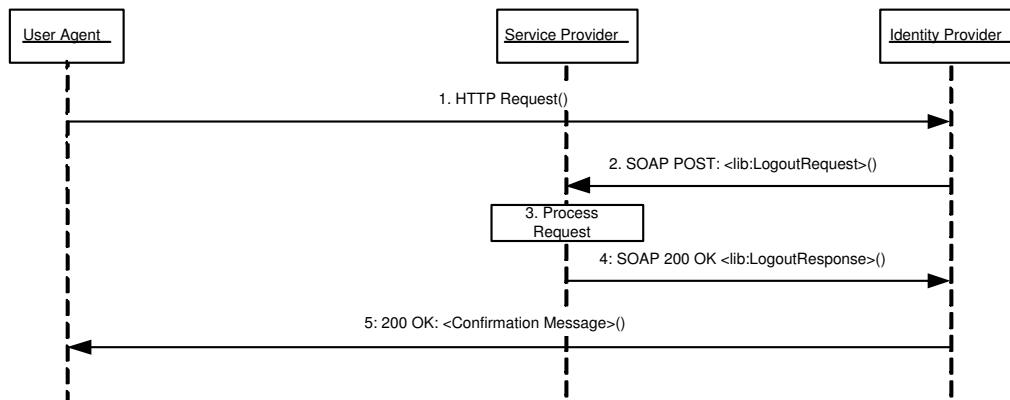
1729 3.5.1.2. SOAP/HTTP-Based Profile

1730 The SOAP/HTTP-based profile uses SOAP over HTTP messaging to communicate a logout request to each service
1731 provider for which the identity provider has provided authentication assertions during the Principal's current session if
1732 the service provider indicated a preference to receive logout request via the SOAP/HTTP-based profile. See Figure 14.

1733 The following URI-based identifier MUST be used when referencing this specific profile:

1734 URI: `http://projectliberty.org/profiles/slo-idp-soap`

1735 This URI identifier MUST be specified in the service provider metadata element `SingleLogoutProtocolProfile` when
1736 the service provider intends to indicate to the identity provider a preference for receiving logout requests via SOAP
1737 over HTTP.



1738

Figure 14. SOAP/HTTP-based profile for single logout initiated at identity provider

1739

1740 Note:

1741 Steps 2 through 4 may be an iterative process for each service provider that has been issued authentication
1742 assertions during the Principal's current session and has indicated a preference to receive logout requests via
1743 the SOAP/HTTP message profile.

1744 3.5.1.2.1. Step 1: Accessing the Single Logout Service

1745 In step 1, the user agent accesses the single logout service URL at the identity provider via an HTTP request.

1746 3.5.1.2.2. Step 2: Logout Request

1747 In step 2, the identity provider sends a SOAP over HTTP request to the SOAP endpoint of each service provider
1748 for which it provided authentication assertions during the Principal's current session. The SOAP message MUST

1749 contain exactly one `<lib:LogoutRequest>` element in the SOAP body and adhere to the construction rules defined
1750 in [\[LibertyProtSchema\]](#).

1751 If a SOAP fault occurs, the identity provider SHOULD employ best efforts to resolve the fault condition and resend
1752 the single logout request to the service provider.

1753 **3.5.1.2.3. Step 3: Processing the Logout Request**

1754 In step 3, the service provider MUST process the `<lib:LogoutRequest>` according to the rules defined in
1755 [\[LibertyProtSchema\]](#).

1756 The service provider MUST invalidate the session for the Principal specified by the name identifier provided by the
1757 identity provider in the `<lib:LogoutRequest>`.

1758 **3.5.1.2.4. Step 4: Responding to the Request**

1759 In step 4, the service provider MUST respond to the `<lib:LogoutRequest>` with a SOAP 200 OK
1760 `<lib:LogoutResponse>` message.

1761 **3.5.1.2.5. Step 5: Confirmation**

1762 In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout has completed.

1763 **3.5.2. Single Logout Initiated at Service Provider**

1764 The profiles in [Section 3.5.2.1](#) and [Section 3.5.2.2](#) are specific to the Principal' initiation of the single logout request
1765 process at the service provider.

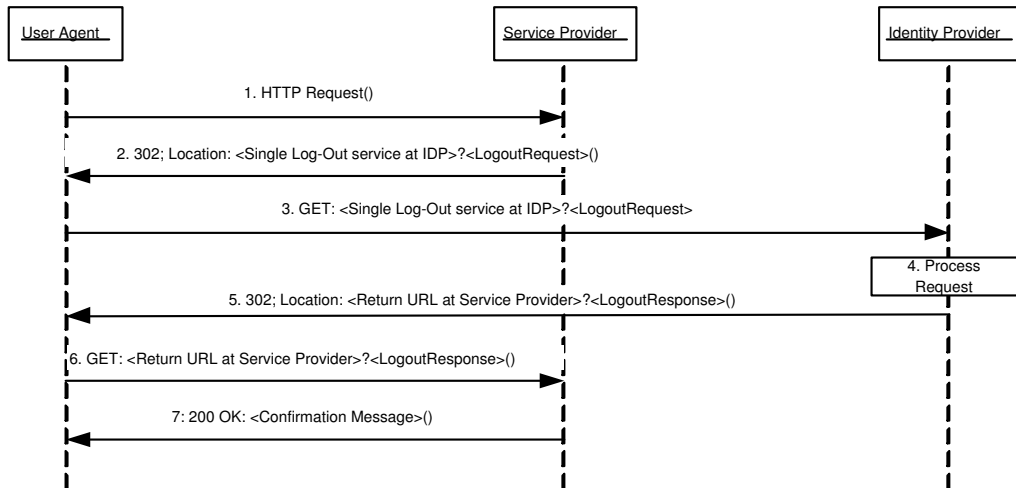
1766 **3.5.2.1. HTTP-Based Profile**

1767 The HTTP-redirect-based profile relies on using a HTTP 302 redirect to communicate a logout request with the identity
1768 provider. The identity provider will then communicate a logout request to each service provider with which it has
1769 established a session for the Principal using the service provider' preferred profile for logout request from the identity
1770 provider (see [Section 3.5.1](#)). See [Figure 15](#).

1771 The following URI-based identifier MUST be used when referencing this specific profile:

1772 URI: `http://projectliberty.org/profiles/slo-sp-http`

1773 This URI identifier is intended for service provider consumption and is not needed in provider metadata.



1774

1775

Figure 15. HTTP-redirect-based profile for single logout initiated at service provider

1776 **Note:**

1777 Step 4 may involve an iterative process by the identity provider to implement the preferred profile for logout
 1778 requests for each service provider that has been issued authentication assertions during the Principal's current
 1779 session.

1780 **3.5.2.1.1. Step 1: Accessing the Single Logout Service at the Service Provider**

1781 In step 1, the user agent accesses the single logout service URL at the service provider indicating that session logout
 1782 is desired at the associated identity provider and all service providers for which this identity provider has provided
 1783 authentication assertions during the Principal's current session. If a current session exists for the Principal at the
 1784 service provider, it is RECOMMENDED that the service provider terminate that session prior to step 2.

1785 **3.5.2.1.2. Step 2: Redirecting to the Single Logout Service at the Identity Provider**

1786 In step 2, the service provider's single logout service responds and redirects the user agent to the single logout service
 1787 URL at the identity provider.

1788 The redirection MUST adhere to the following rules:

- 1789 • The Location HTTP header MUST be set to the identity provider's single logout service URL.
- 1790 • The identity provider's single logout service URL MUST specify https as the URL scheme; if another scheme is
 1791 specified, the service provider MUST NOT redirect to the identity provider.
- 1792 • The Location HTTP header MUST include a <query> component containing the <lib:LogoutRequest>
 1793 protocol message as defined in [LibertyProtSchema] with formatting as specified in 3.1.2.

1794 The HTTP response MUST take the following form:

```
1795  
1796 <HTTP-Version> 302 <Reason Phrase>  
1797 <other headers>  
1798 Location : https://<Identity Provider single log-out service URL>?<query>  
1799 <other HTTP 1.0 or 1.1 components>  
1800
```

1801 where:

1802 <Identity Provider single log-out service URL>

1803 This element provides the host name, port number, and path components of the single logout service URL at the
1804 identity provider.

1805 <query>= ...<URL-encoded LogoutRequest>...

1806 The <query> MUST contain a single logout request.

1807 **3.5.2.1.3. Step 3: Accessing the Identity Provider Single Logout Service**

1808 In step 3, the user agent accesses the identity provider's single logout service URL with the <lib:LogoutRequest>
1809 information attached to the URL fulfilling the redirect request.

1810 **3.5.2.1.4. Step 4: Processing the Request**

1811 In step 4, the identity provider MUST process the <lib:LogoutRequest> according to the rules defined in
1812 [\[LibertyProtSchema\]](#).

1813 Each service provider for which the identity provider has provided authentication assertions during the Principal's
1814 current session MUST be notified via the service provider's preferred profile for logout request from the identity
1815 provider (see [Section 3.5.1](#)).

1816 The identity provider's current session with the Principal MUST be terminated, and no more authentication assertions
1817 for the Principal are to be given to service providers.

1818 **3.5.2.1.5. Step 5: Redirecting to the Service Provider Return URL**

1819 In step 5, the identity provider's single logout service responds and redirects the user agent back to service provider
1820 using the return URL location obtained from the SingleLogoutServiceReturnURL metadata element. If the URL-
1821 encoded <lib:LogoutRequest> message received in step 3 contains a parameter named RelayState, then the identity
1822 provider MUST include a <query> component containing the same RelayState parameter and its value in its response
1823 to the service provider.

1824 The purpose of this redirect is to return the user agent to the service provider.

1825 The HTTP response MUST take the following form:

```
1826  
1827 <HTTP-Version> 302 <Reason Phrase>  
1828 <other headers>  
1829 Location : https://<Service Provider Return Service URL>?<query>  
1830 <other HTTP 1.0 or 1.1 components>  
1831
```

1832 where:

1833 <Service Provider Service Return URL>

1834 This element provides the host name, port number, and path components of the return URL location at the service
 1835 provider.

1836 `<query>= ...<URL-encoded LogoutResponse>`

1837 The `<query>` component **MUST** contain a single logout response. The `<URL-encoded LogoutResponse>` com-
 1838 ponent **MUST** contain the identical RelayState parameter and its value that was received in the URL-encoded logout
 1839 request message obtained in step 3. If no RelayState parameter was provided in the step 3 message, then a RelayState
 1840 parameter **MUST NOT** be specified in the `<URL-encoded LogoutResponse>`.

1841 **3.5.2.1.6. Step 6: Accessing the Service Provider Return URL**

1842 In step 6, the user agent accesses the service provider’s return URL location fulfilling the redirect request.

1843 **3.5.2.1.7. Step 7: Confirmation**

1844 In step 7, the user agent is sent an HTTP response that confirms the requested action of a single logout has been
 1845 completed.

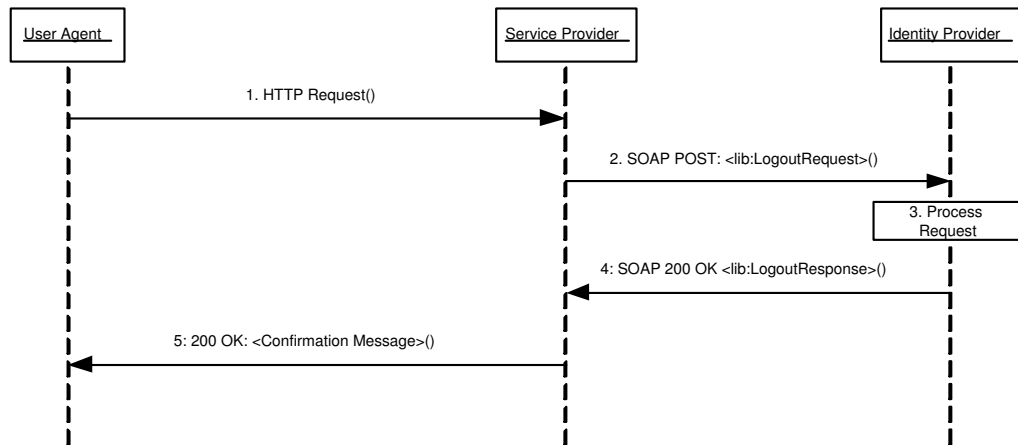
1846 **3.5.2.2. SOAP/HTTP-Based Profile**

1847 The SOAP/HTTP-based profile relies on using SOAP over HTTP messages to communicate a logout request to
 1848 the identity provider. The identity provider will then communicate a logout request to each service provider it has
 1849 established a session with for the Principal via the service provider’ preferred profile for logout requests from the
 1850 identity provider (see [Section 3.5.1](#)). See [Figure 16](#).

1851 The following URI-based identifier **MUST** be used when referencing this specific profile:

1852 URI: `http://projectliberty.org/profiles/slo-sp-soap`

1853 This URI identifier is intended for service provider consumption and is not needed in provider metadata.



1854

Figure 16. SOAP/HTTP-based profile for single logout initiated at service provider

1855

1856 **Note:**

1857 Step 3 may involve an iterative process by the identity provider to implement the preferred profile for logout
 1858 requests for each service provider that has been issued authentication assertions during the Principal’s current
 1859 session.

1860 **3.5.2.2.1. Step 1: Accessing Single Logout Service**

1861 In step 1, the user agent accesses the single logout service URL at the service provider via an HTTP request.

1862 **3.5.2.2.2. Step 2: Logout Request**

1863 In step 2, the service provider sends a SOAP over HTTP request to the identity provider's SOAP endpoint. The
1864 SOAP message MUST contain exactly one `<lib:LogoutRequest>` element in the SOAP body and adhere to the
1865 construction rules as defined in [\[LibertyProtSchema\]](#).

1866 If a SOAP fault occurs, the service provider SHOULD employ best efforts to resolve the fault condition and resend
1867 the single logout request to the identity provider.

1868 **3.5.2.2.3. Step 3: Processing the Logout Request**

1869 In step 3, the identity provider MUST process the `<lib:LogoutRequest>` according to the rules defined in
1870 [\[LibertyProtSchema\]](#).

1871 Each service provider for which the identity provider has provided authentication assertions during the Principal's
1872 current session MUST be requested to logout the Principal via the service provider's preferred profile for logout
1873 requests from the identity provider. If the identity provider determines that one or more of service providers to which
1874 it has provided assertions regarding this Principal do not support the SOAP profiles for the single logout, the identity
1875 provider MUST return a `<lib:LogoutResponse>` containing a status code of `<lib:UnsupportedProfile>`. The
1876 service provider MUST then re-submit its `LogoutRequest` via the HTTP profile described above.

1877 The identity provider's current session with the Principal MUST be terminated, and no more authentication assertions
1878 for the Principal are to be given to service providers.

1879 **3.5.2.2.4. Step 4: Responding to the Logout Request**

1880 In step 4, the identity provider MUST respond to the `<lib:LogoutRequest>` with a SOAP 200 OK
1881 `<lib:LogoutResponse>` message.

1882 **3.5.2.2.5. Step 5: Confirmation**

1883 In step 5, the user agent is sent an HTTP response that confirms the requested action of single logout was completed.

1884 **3.6. Identity Provider Introduction**

1885 This section defines the profiles by which a service provider discovers which identity providers a Principal is using.
1886 In identity federation networks having more than one identity provider, service providers need a means to discover
1887 which identity providers a Principal uses. The introduction profile relies on a cookie that is written in a domain that
1888 is common between identity providers and service providers in an identity federation network. The domain that the
1889 identity federation network predetermines for a deployment is known as the common domain in this specification, and
1890 the cookie containing the list of identity providers is known as the common domain cookie.

1891 Implementation of this profile is OPTIONAL. Whether identity providers and service providers implement this profile
1892 is a policy and deployment issue outside the scope of this specification. Also, which entities host web servers in the
1893 common domain is a deployment issue and is outside the scope of this specification.

1894 **3.6.1. Common Domain Cookie**

1895 The name of the cookie MUST be `_liberty_idp`. The format of the cookie content MUST be a list of base64-encoded
1896 (see [\[RFC2045\]](#)) identity provider succinct IDs separated by a single white space character. The identity provider IDs
1897 MUST adhere to the creation rules as defined in [Section 3.2.2.2](#). The identity provider ID is a metadata element, as
1898 defined in [\[LibertyMetadata\]](#).

1899 The common domain cookie writing service SHOULD append the identity provider ID to the list. If the identity
1900 provider ID is already present in the list, it MAY remove and append it when authentication of the Principal occurs.
1901 The intent is that the most recently established identity provider session is the last one in the list.

1902 The cookie MUST be set with no Path prefix or a Path prefix of "/". The Domain MUST be set to "[common-domain]"
1903 where [common-domain] is the common domain established within the identity federation network for use with the
1904 introduction protocol. The cookie MUST be marked as Secure.

1905 The cookies MAY be either session or persistent (see [\[RFC2109\]](#)), the implementation of which is a policy and
1906 deployment issue of the identity federation network.

1907 **3.6.2. Setting the Common Domain Cookie**

1908 After the identity provider authenticates a Principal, it MAY set the common domain cookie. The means by which
1909 the identity provider sets the cookie are implementation-specific so long as the cookie is successfully set with the
1910 parameters given above. One possible implementation strategy follows and should be considered non-normative. The
1911 identity provider may:

- 1912 • Have previously established a DNS and IP alias for itself in the common domain
- 1913 • Redirect the user agent to itself using the DNS alias using a URL specifying "https" as the URL scheme. The
1914 structure of the URL is private to the implementation and may include session information needed to identify the
1915 user-agent.
- 1916 • Redirect the user agent to itself using the DNS alias using a URL specifying "https" as the URL scheme. The
1917 structure of the URL is private to the implementation and may include session information needed to identify the
1918 user-agent.
- 1919 • Redirect the user agent back to itself, or, if appropriate, to the service provider.

1920 **3.6.3. Obtaining the Common Domain Cookie**

1921 When a service provider needs to discover which identity providers the Principal uses, it invokes a protocol exchange
1922 designed to present the common domain cookie to the service provider after it is read by an HTTP server in the
1923 common domain.

1924 If the HTTP server in the common domain is operated by the service provider, the service provider MAY redirect the
1925 user agent to an identity provider's intersite transfer service for an optimized single sign-on process.

1926 The specific means by which the service provider reads the cookie are implementation-specific so long as it is able
1927 to cause the user agent to present cookies that have been set with the parameters given in section [Section 3.6.1](#). One
1928 possible implementation strategy is described as follows and should be considered non-normative. Additionally, it
1929 may be sub-optimal for some applications.

- 1930 • Have previously established a DNS and IP alias for itself in the common domain
- 1931 • Redirect the user agent to itself using the DNS alias using a URL specifying "https" as the URL scheme. The
1932 structure of the URL is private to the implementation and may include session information needed to identify the
1933 user-agent.
- 1934 • Set the cookie on the redirected user agent using the parameters specified above
- 1935 • Redirect the user agent back to itself, or, if appropriate, to the service provider.

1936 3.7. NameIdentifier Mapping Profile

1937 The NameIdentifier mapping profile specifies how a service provider may obtain a NameIdentifier for a Principal it
1938 has federated in the "namespace" of a different service provider, by querying an identity provider that has federated
1939 the Principal with both service providers. This NameIdentifier may be used to obtain additional information about a
1940 Principal from a SAML authority, or used for other non-specific purposes. In most cases, the encryption profile in the
1941 following section will be used to obfuscate and time-limit this identifier to restrict its use.

1942 The NameIdentifier mapping profile makes use of the following metadata elements, as defined in [\[LibertyMetadata\]](#):

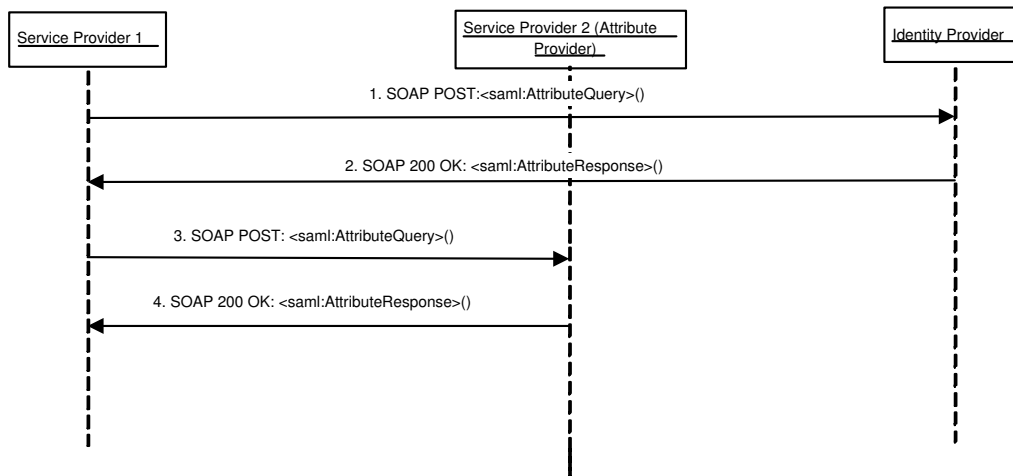
- 1943 • NameIdentifierMappingProtocolProfile - A URI indicating the profile of this protocol supported by the
1944 identity provider.
- 1945 • SOAPEndpoint - The SOAP endpoint location at the identity provider to which Liberty SOAP messages are sent.

1946 3.7.1. SOAP-based NameIdentifier Mapping

1947 The SOAP-based profile relies on a SOAP request and response to query for and return the NameIdentifier. A
1948 requesting service provider issues a SOAP request to an identity provider, requesting a different NameIdentifier for a
1949 named Principal in the namespace of a service provider or affiliation group. This NameIdentifier may then be used
1950 to query another Liberty provider offering SAML services for additional information about the named Principal. See
1951 [Figure 17](#).

1952 The following URI-based identifier MUST be used when referencing this specific profile:

1953 URI: <http://projectliberty.org/profiles/nim-sp-http>



1954

1955 Figure 17. SOAP-based profile for name identifier mapping

1956 3.7.1.1. Step 1: Issuance of Request

1957 In step 1, the service provider sends a SOAP over HTTP request to the SOAP endpoint of the identity provider it is
1958 querying. The SOAP message MUST contain exactly one `<lib:NameIdentifierMappingRequest>` element in
1959 the SOAP body and adhere to the construction rules defined in [\[LibertyProtSchema\]](#).

1960 3.7.1.2. Step 2: Processing the Request

1961 In step 2, the identity provider MUST process the `<lib:NameIdentifierMappingRequest>` according to the rules
1962 defined in [\[LibertyProtSchema\]](#).

1963 The identity provider is NOT required to honor the request for a mapped NameIdentifier, but it MUST respond to the
1964 request with an appropriate status.

1965 **3.7.1.3. Step 3: Responding to the Request**

1966 In step 3, the identity provider MUST respond to the `<lib:NameIdentifierMappingRequest>` with a SOAP 200
1967 OK `<lib:NameIdentifierMappingResponse>` message.

1968 **3.7.1.4. Step 4: Requesting SAML attributes using a mapped NameIdentifier**

1969 Note: This step is not normatively specified by Liberty, and is shown only for illustrative purposes. The requesting
1970 service provider may use the mapped NameIdentifier of the Principal to issue a `<saml:AttributeQuery>`. This
1971 MUST adhere to the rules specified in [\[SAMLCore11\]](#)

1972 **3.7.1.5. Step 5: Returning a `<saml:AttributeStatement>`**

1973 Note: This step is not normatively specified by Liberty, and is shown only for illustrative purposes. A service provider
1974 receiving a `<saml:AttributeQuery>` may return a `<saml:AttributeStatement>`. This action MUST conform
1975 to the rules specified in [\[SAMLCore11\]](#).

1976 **3.7.1.6. Security Considerations**

1977 In addition to the usual considerations relating to Liberty and SAML protocols (see [\[SAMLCore11\]](#)), an identity
1978 provider SHOULD encrypt or otherwise obfuscate the NameIdentifier returned to the requesting service provider, so
1979 that it is opaque and of time-limited use to the requester. A way of accomplishing this is described in the next section.

1980 **3.8. NameIdentifier Encryption Profile**

1981 The Liberty NameIdentifier encryption profile allows a principal's NameIdentifier to be encrypted such that only
1982 the identity or service provider possessing the decryption key can deduce the identity of the principal when the
1983 NameIdentifier is included in a SAML or Liberty protocol message. The identifier is encrypted in such a fashion
1984 that it may only be used for a limited time, preventing correlation of the encrypted value across multiple logical
1985 transactions.

1986 The NameIdentifier encryption profile make use of the following metadata element, as defined in [\[LibertyMetadata\]](#):

- 1987 • `KeyDescriptor` - Defines a public key to use when wrapping the keys used in encrypting data for a provider (the
1988 key-encrypting key)

1989 **3.8.1. XML Encryption-based NameIdentifier Encryption**

1990 The XML Encryption-based profile relies on the use of [\[xmenc-core\]](#) to format and encode the resulting encrypted
1991 identifier and possibly the wrapped encryption key.

1992 The following URI-based identifier MUST be used when referencing this specific profile:

1993 URI: urn:liberty:iff:nameid:encrypted

1994 **3.8.1.1. Step 1: Encrypting and encoding a NameIdentifier value.**

1995 The encrypting provider first transforms the original `<saml:NameIdentifier>` element into a
1996 `<EncryptableNameIdentifier>` element, which is an extension of the original element. The `NameQualifier`,
1997 `IssueInstant`, and `Nonce` attributes are set as defined by [\[LibertyProtSchema\]](#).

1998 If not already generated for the target provider, an encryption key is generated and is then itself encrypted with the
1999 key specified in the target provider's `<KeyDescriptor>` metadata element with a `use` attribute of `encryption`, or
2000 with a predefined key exchanged out of band. The wrapped encryption key is placed into a `<xenc:EncryptedKey>`
2001 element.

2002 If the symmetric encryption key is not included, because it has been exchanged out of band, and/or is being reused,
2003 then the encrypting provider MUST include additional information in the `<xenc:EncryptedData>` element that
2004 indicates to the target provider which decryption key to use in decrypting the identifier. This information MUST be
2005 sufficient to identify the key to use without the target knowing the encrypting provider's identity before decryption
2006 occurs.

2007 The encryption key is then applied to the `<EncryptableNameIdentifier>` element, producing an
2008 `<xenc:EncryptedData>` element with a `Type` of `http://www.w3.org/2001/04/xmenc#Element`.

2009 The resulting `<xenc:EncryptedData>` element, and optionally the `<xenc:EncryptedKey>` element, are then
2010 enclosed in an `<EncryptedNameIdentifier>` element. The element is base-64 encoded and the result is placed
2011 into a `<saml:NameIdentifier>` whose `Format` attribute MUST be `urn:liberty:iff:nameid:encrypted`.

2012 **3.8.1.2. Step 2: Decoding and decrypting a NameIdentifier value.**

2013 The decrypting provider first decodes the base-64 encoded data and recovers the `<EncryptedNameIdentifier>`
2014 element.

2015 The `<xenc:EncryptedData>` and optional `<xenc:EncryptedKey>` elements are then used to recover the symmetric
2016 encryption key and algorithm and decrypt the `<EncryptableNameIdentifier>` element.

2017 The provider can then examine the attributes to determine the identity federation to which the name identifier applies
2018 and insure that the time of issue of the encrypted identifier and the nonce value do not preclude the use of the identifier
2019 for the intended purpose.

2020 **3.8.2. Security Considerations**

2021 The profile is designed to meet the needs of providers in addressing the security considerations of other profiles, such
2022 as the NameIdentifier Mapping Profile in the previous section. To insure the integrity of this profile, either symmetric
2023 encryption keys MUST NOT be reused, or if they are, then symmetric encryption keys MUST be reused between
2024 different principals federated with a given provider and MUST NOT be reused between different providers. It is
2025 RECOMMENDED that symmetric encryption keys, if reused, be renewed periodically. Furthermore, reuse of keys
2026 REQUIRES that a chaining mode with a unique initialization vector generated per encryption be used.

2027 Receiving providers MUST take care to use the `IssueInstant` and `Nonce` attributes to prevent replay and long term
2028 use of the same encrypted identifier.

References

2029

Normative

2030

- 2031 [HTML4] Raggett, D., Le Hors, A., Jacobs, I, eds. (24 December 1999). "HTML 4.01 Specification ," Recommendation, W3C <http://www.w3.org/TR/html401/> [August 2003].
- 2032
- 2033 [LibertyAuthnContext] Madsen, Paul , eds. "Liberty ID-FF Authentication Context Specification," Version 1.2, Liberty Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 2034
- 2035 [LibertyGlossary] Wason, Thomas, eds. "Liberty Technical Glossary," Version 1.2, Liberty Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 2036
- 2037 [LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 1.0, Liberty Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 2038
- 2039 [LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version 1.2, Liberty Alliance Project (12 November 2003). <http://www.projectliberty.org/specs>
- 2040
- 2041 [SAMLBind11] Maler, E., Mishra, P., Philpott, R., eds. (27 May 2003). "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards http://www.oasis-open.org/committees/documents.php?wg_abbrev=security
- 2042
- 2043
- 2044
- 2045 [SAMLCore11] Maler, E., Mishra, P., Philpott, R., eds. (27 May 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards http://www.oasis-open.org/committees/documents.php?wg_abbrev=security
- 2046
- 2047
- 2048
- 2049 [Schema1] Thompson, H.S., Beech, D., Maloney, M., Mendleson, N., eds. (May 2002). "XML Schema Part 1: Structures," Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlschema-1/>
- 2050
- 2051 [SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Thatte, Satish, Winer, Dave, eds. World Wide Web Consortium W3C Note (08 May 2000). <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 2052
- 2053
- 2054
- 2055 [SSL] Frier, A., Karlton, P., Kocher, P., eds. (November 1996). Netscape Communications Corporation "The SSL 3.0 Protocol,"
- 2056
- 2057 [RFC1750] Eastlake , D., Crocker, S., Schiller, J., eds. (December 1994). "Randomness Recommendations for Security ," RFC 1750, Internet Engineering Task Force <http://www.rfc-editor.org/rfc/rfc1750.txt> [August 2003].
- 2058
- 2059
- 2060 [RFC1945] Berners-Lee, T., Fielding, R., Frystyk, H., eds. (May 1996). "Hypertext Transfer Protocol – HTTP/1.0 ," RFC 1945, Internet Engineering Task Force <http://www.rfc-editor.org/rfc/rfc1945.txt> [August 2003].
- 2061
- 2062 [RFC2045] Freed, N., Borenstein, N., eds. (November 1996). "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies ," RFC 2045., Internet Engineering Task Force <http://www.rfc-editor.org/rfc/rfc2045.txt> [http://www.rfc-editor.org/rfc/rfc2045.txt] [November 1996]
- 2063
- 2064
- 2065 [RFC2109] Kristol, D., Montulli, L., eds. (February 1997). "HTTP State Management Mechanism ," RFC 2109 [Obsoleted by RFC2965], Internet Engineering Task Force <http://www.rfc-editor.org/rfc/rfc2109.txt> [August 2003].
- 2066
- 2067
- 2068 [RFC2119] Bradner, S., eds. "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet Engineering Task Force (March 1997). <ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt>
- 2069

- 2070 [RFC2246] Dierks, T., Allen, C., , , , eds. (January 1999). "The TLS Protocol," Version 1.0 RFC 2246, Internet
2071 Engineering Task Force <http://www.rfc-editor.org/rfc/rfc2246.txt> [20 December 2002].
- 2072 [RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI):
2073 Generic Syntax," RFC 2396, The Internet Engineering Task Force <http://www.rfc-editor.org/rfc/rfc2396.txt>
2074 [18 December 2002].
- 2075 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June 1999).
2076 "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force [http://www.rfc-](http://www.rfc-editor.org/rfc/rfc2616.txt)
2077 [editor.org/rfc/rfc2616.txt](http://www.rfc-editor.org/rfc/rfc2616.txt) [18 December 2002].
- 2078 [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Stewart, L., eds. (June 1999). "HTTP
2079 Authentication: Basic and Digest Access Authentication," RFC 2617, The Internet Engineering Task Force
2080 <http://www.rfc-editor.org/rfc/rfc2617.txt> [18 December 2002].
- 2081 [RFC3106] Eastlake, D., Goldstein, T., eds. (April 2001). "ECML v1.1: Field Specifications for E-Commerce ," RFC
2082 3106, Internet Engineering Task Force <http://www.rfc-editor.org/rfc/rfc1750.txt> [August 2003].
- 2083 [WML] none stated, , eds. (2000). "Wireless Application Protocol: Wireless Markup Language Specification,"
2084 Version 1.3, Open Mobile Alliance [http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-](http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf)
2085 [a.pdf](http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf) [November 2003].
- 2086 [xmlenc-core] Eastlake, Donald, Reagle, Joseph, eds. (December 2002). "XML Encryption Syntax and Processing,"
2087 W3C Recommendation, World Wide Web Consortium <http://www.w3.org/TR/xmlenc-core/>
- 2088 **Informative**
- 2089 [LibertyImplGuide] Kemp, John, eds. "Liberty ID-FF Implementation Guidelines," Version 1.2, Liberty Alliance
2090 Project (). <http://www.projectliberty.org/specs>