



1

Liberty Bindings and Profiles Specification

2

Version 1.1

3

15 January 2003

4

5 Document Description: **liberty-architecture-bindings-profiles-v1.1**

6 **Notice**

7 Copyright © 2002, 2003 ActivCard; American Express Travel Related Services; America Online,
8 Inc.; Bank of America; Bell Canada; Catavault; Cingular Wireless; Cisco Systems, Inc.; Citigroup;
9 Communicator, Inc.; Consignia; Cyberun Corporation; Deloitte & Touche LLP; Earthlink, Inc.;
10 Electronic Data Systems, Inc.; Entrust, Inc.; Ericsson; Fidelity Investments; France Telecom;
11 Gemplus; General Motors; Hewlett-Packard Company; i2 Technologies, Inc.; Intuit Inc.;
12 MasterCard International; NEC Corporation; Netegrity; NeuStar; Nextel Communications; Nippon
13 Telegraph and Telephone Company; Nokia Corporation; Novell, Inc.; NTT DoCoMo, Inc.;
14 OneName Corporation; Openwave Systems Inc.; PricewaterhouseCoopers LLP; Register.com;
15 RSA Security Inc; Sabre Holdings Corporation; SAP AG; SchlumbergerSema; SK Telecom; Sony
16 Corporation; Sun Microsystems, Inc.; United Airlines; VeriSign, Inc.; Visa International;
17 Vodafone Group Plc; Wave Systems;. All rights reserved.

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

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

34 Liberty Alliance Project
35 Licensing Administrator
36 c/o IEEE-ISTO
37 445 Hoes Lane
38 Piscataway, NJ 08855-1331, USA
39 info@projectliberty.org

40

41 **Editors**

42 Jason Rouault, Hewlett-Packard Company
43 Tom Wason. IEEE-ISTO

44 **Contributors**

45

ActivCard	NEC Corporation
American Express Travel Related Services	Netegrity
America Online, Inc.	NeuStar
Bank of America	Nextel Communications
Bell Canada	Nippon Telegraph and Telephone Company
Catavault	Nokia Corporation
Cingular Wireless	Novell, Inc.
Cisco Systems, Inc.	NTT DoCoMo, Inc.
Citigroup	OneName Corporation
Communicator, Inc.	Openwave Systems Inc.
Consignia	PricewaterhouseCoopers LLP
Cyberun Corporation	Register.com
Deloitte & Touche LLP	RSA Security Inc
EarthLink, Inc.	Sabre Holdings Corporation
Electronic Data Systems, Inc.	SAP AG
Entrust, Inc.	SchlumbergerSema
Ericsson	SK Telecom
Fidelity Investments	Sony Corporation
France Telecom	Sun Microsystems, Inc.
Gemplus	United Airlines
General Motors	VeriSign, Inc.
Hewlett-Packard Company	Visa International
i2 Technologies, Inc.	Vodafone Group Plc
Intuit Inc.	Wave Systems
MasterCard International	

46

47 **Document History**

Version #	Date	Editor	Scope of changes
1.0	14-Mar-02	Jason Rouault	Initial Draft Based on Liberty V1.0
1.1	04-Nov-02	Jason Rouault, Tom Wason	<p>CR1097: Typo corrections.</p> <p>CR1098: collapse the HTTP-GET-Based and HTTP-Redirect-Based profiles into the same profile</p> <p>CR1106: prescriptive alternative for countermeasure to Unsigned <code><lib:AuthnRequest></code> message. Section 4.4.2.1.</p> <p>CR1118: Added rogue participant threat notice. Section 4.4.2.3.</p> <p>CR1122: Describe rogue impersonation threat and countermeasure. Section 4.4.2.1.</p> <p>CR1130: Resolution of LEC vulnerability to a man-in-the-middle attack. Section 3.2.5.1.</p> <p>CR1138: Changed namespace notation <code>"http://projectliberty.org/schemas/core/2002/08"</code> to <code>"http://projectliberty.org/schemas/core/2002/12"</code>. Section 1.1, Section 3.2.5.2 step 3.</p> <p>CR1139: Change cookie constraints. In Section 3.6.1 changed “identity provider ID” to “identity provider succinct ID”. Restricted Path prefix. In Section 3.6.2 removed figure and steps, added text describing setting common domain cookie. In section 3.6.3 removed steps and figure, inserted text describing reading the common domain cookie.</p> <p>CR1140: Clarification of construction of ProviderSuccinctID. Section 3.2.2.2</p> <p>CR1147: Correct typo.</p> <p>CR1148: Remove use of element <code>samlp:RespondWith</code>. Section 3.1.2.1.2.</p> <p>CR1150: Added note on escaping base64-encoded signature values. Section 3.1.2.1.</p> <p>CR1151: Added “The service URL provided by the provider (the URL to which <code><query></code> parameters are added) MUST NOT contain any pre-existing <code><query></code> parameter values.” Section 3.1.2.1</p>

			<p>CR1156: URI specified in the service provider metadata element FederationTerminationNotificationProtocolProfile. Section 3.4.1.</p> <p>CR1157: Session persistence at federation termination notification. Section 3.4.2</p> <p>CR1159: Changed "...SAML artifact on success or empty on failure." To "...SAML artifact on success or on failure." Added following sentence: "In the case of failure, the status information will be communicated in the <samlp:Response> returned." in step 9. Section 3.2.2.1</p> <p>CR1160, CR1161: Generalizing Register Name Identifier Profile. Register name identifier profile is changed to support NameIdentifier changes by both Identity Provider and Service Provider. Section 3.3. Change Name Registration Profile description to reflect the bi-directional nature of the profile for establishing or changing a Principal's name identifier. Section 3.</p> <p>CR1162: Change "Name" to "NameIdentifier". Sections 3.1.2.1.3, 3.1.2.1.4</p> <p>CR1166: Changed example to provide correct content type. Section 2.2.</p> <p>CR1167: Change header from "Liberty-LECP" to "Liberty-Enabled". Section 3.2.5.1.</p> <p>CR1168: Delete requirement for base64 encoding in Section 3.2.5.2, Step 3.</p> <p>CR1169: Change "...transfer service responds and redirects..." to "...transfer service responds and sends...", delete redirection rules from Step 3. Added, "The form and contents of the HTTP response in this step are profile-dependent." in following line. Removed "attached to the URL fulfilling the redirect request" from step 4. Section 3.2.1. Added step 3 to Sections 3.2.2.1 and 3.2.3.</p> <p>CR1178: Changed "Maximum URL length of 256 bytes." To "Minimum maximum URL length of 256 bytes." Section 3.1.1.</p> <p>CR1180: Modified Step 5: Processing <AuthnRequest>, Section 3.2.1.</p> <p>CR1181: Modified Step 6: HTTP Response with <AuthnResponse> or Artifact, Section</p>
--	--	--	--

			<p>3.2.1.</p> <p>CR1182: Changed common requirements 3, 4 &6, deleted #12. Section 3.1.</p> <p>CR1219: Changed references to Name Registration Profiles, now plural. Sections 3, 3.3.</p> <p>CR1220: Specify algorithms and identifiers, Section 3.1.2.1</p> <p>CR1221: Added encodings for RegisterNameIdentifierRequest, Section 3.1.2.1.6 and RegisterNameIdentifierResponse, Section 3.1.2.1.7.</p> <p>CR1223: Correct typos. Sections 3.2.1, 3.3, 3.5, 3.5.2.2.</p> <p>CR1224: Replaced <lib:Assertion> with <saml:Assertion> 8 places.</p> <p>CR1225: Corrected contents of <saml:assertion>, Section 3.2.1, Step 9.</p> <p>CR1228: Removed “asynchronous”, Section 3.3.2.2.</p> <p>CR1229: Revised code for the original <lib:AuthnRequest> message, Section 3.1.2.1.2.</p> <p>CR1230: Changed Liberty-LECP to Liberty-Enabled, 2 places in Section 3.2.5.1.</p> <p>CR1231: Added URI-based identifiers to Register Name Identifier profiles, Sections 3.3.2.1, 3.3.2.2</p> <p>CR1232: Typo</p> <p>CR1233: Added 2 metadata elements to Register Name Identifier profiles, Section 3.3.</p> <p>CR1242: Corrected error in implementing CR 1181, Steps 6 in both Sections 3.2.5.2 and 3.2.1.</p> <p>CR1243: Replace RELAYSTATE and LRURL with RelayState throughout.</p> <p>CR1244: Removed references to ProviderSuccinctID in Section 3.2.2.2.</p> <p>CR1245: RegisterNameIdentifierReturnURL is changed to RegisterNameIdentifierServiceReturnURL, Section 3.3.</p> <p>CR1253: Changed all instances of</p>
--	--	--	--

			<p>LogoutNotification to LogoutRequest, Section 3.5.</p> <p>CR1255: Revised Single Logout LogoutRequest profiles to be similar to other requests by adding LogoutResponse. Substantive changes to all of Section 3.5</p> <p>CR1256: Correct element order in XML example, Section 3.2.5.2, Step 3.</p> <p>CR1259: Change AuthnContextComparisonType to AuthnContextComparison, Section 3.1.2.1.2.</p> <p>CR1260: Remove extraneous RegisterNameIdentifierReturnURL, Section 3.3.</p> <p>CR1261: Added LogoutResponse encoding, Section 3.1.2.1.5.</p> <p>CR1262: Added mechanism for extending the sampl:Request type in Section 3.2.1. SOAP example modified in Section 2.2.</p> <p>CR1263: Editorial adjustments.</p>
1.1 - 18	12-Dec-02	Jason Rouault, Tom Wason	<p>CR1179: Divided references into normative and informative categories, Section 5.</p> <p>CR1264: Add RelayState to LogoutRequest example Section 3.1.2.1.4.</p> <p>CR1266: See CR1282.</p> <p>CR1268: Replaced SOAP over HTTP example. Section 2.2.</p> <p>CR1269: Change OLDProvidedNameIdentifier to OldProvidedNameIdentifier, Section 3.1.2.1.6.</p> <p>CR1271: Clarify POST error reporting, step 4, Section 3.2.5.2</p> <p>CR1273: Corrected typo.</p> <p>CR1274: Reference “Minimum maximum”. Section 3.1.1.</p> <p>CR1275: Re-entered cookie writing instruction for appending IDP Succinct ID. Section 3.6.1.</p> <p>CR1278: Notes for implementing multilevel status codes. Sections 3.1.2.1, 3.1.2.1.5, 3.1.2.1.7.</p> <p>CR1282: Removed requirement that <query> parameter name MUST be an XML attribute name, Section 3.1.2.1. Differentiated</p>

			<p>NameQualifier values for RegisterNameIdentifierRequest, Section 3.1.2.1.6.</p> <p>CR1283: Added instruction for use of predefined xmlns:prefixes for status codes in URL-encoded responses. Section 3.1.2.1.</p> <p>CR1284: Add RelayState to FederationTerminationNotification URL-Encoding, Section 3.1.2.1.3.</p> <p>CR1285: URL-Encoding templates updated, corrected, consistently formatted Section 3.1.2.1.</p> <p>CR1289: delete note on addition of RelayState element in step 2 of sections 3.3.1.1, 3.4.1.1, 3.5.1.1.1, 3.5.1.1.2 and 3.5.2.1</p>
1.1 Final	15-Jan-03	John Kemp	<p>Corrected SLO profiles for LogoutResponse. Regenerated TOC Typos</p>

49	Table of Contents	
50	1 Introduction	10
51	1.1 Notation	10
52	2 Protocol Bindings.....	11
53	2.1 SOAP Binding for Liberty	11
54	2.2 Example of Message Exchange Using SOAP over HTTP	11
55	3 Profiles.....	13
56	3.1 Common Requirements.....	14
57	3.1.1 User Agent	15
58	3.1.2 Formatting and Encoding of Protocol Messages	15
59	3.1.3 Provider Metadata.....	20
60	3.2 Single Sign-On and Federation Profiles	20
61	3.2.1 Common Interactions and Processing Rules.....	20
62	3.2.2 Liberty Browser Artifact Profile	24
63	3.2.3 Liberty Browser POST Profile.....	28
64	3.2.4 Liberty WML POST Profile.....	30
65	3.2.5 Liberty-Enabled Client and Proxy Profile	33
66	3.3 Register Name Identifier Profiles	38
67	3.3.1 Register Name Identifier Initiated at Identity Provider	39
68	3.3.2 Register Name Identifier Initiated at Service Provider	43
69	3.4 Identity Federation Termination Notification Profiles.....	44
70	3.4.1 Federation Termination Notification Initiated at Identity Provider	45
71	3.4.2 Federation Termination Notification Initiated at Service Provider.....	49
72	3.5 Single Logout Profiles.....	50
73	3.5.1 Single Logout Initiated at Identity Provider	51
74	3.5.2 Single Logout Initiated at Service Provider.....	58
75	3.6 Identity Provider Introduction.....	62
76	3.6.1 Common Domain Cookie.....	62
77	3.6.2 Setting the Common Domain Cookie	63
78	3.6.3 Obtaining the Common Domain Cookie	63
79	4 Security Considerations	63
80	4.1 Introduction	63
81	4.2 General Requirements.....	64
82	4.2.1 Security of SSL and TLS.....	64
83	4.2.2 Security Implementation.....	64
84	4.3 Classification of Threats	64
85	4.3.1 Threat Model.....	64
86	4.3.2 Rogue and Spurious Entities	65
87	4.3.3 Active and Passive Attackers	65
88	4.3.4 Scenarios	65
89	4.4 Threat Scenarios and Countermeasures.....	66
90	4.4.1 Common Threats for All Profiles.....	66
91	4.4.2 Single Sign-On and Federation	67
92	4.4.3 Name Registration	70
93	4.4.4 Federation Termination: HTTP-Redirect-Based Profile	70
94	4.4.5 Single Logout: HTTP-Redirect-Based Profile.....	70
95	4.4.6 Identity Provider Introduction	71
96	5 References	72
97	5.1 Normative	72
98	5.2 Informative	74

99

100 1 Introduction

101 This specification defines the bindings and profiles of the Liberty protocols and messages to
102 HTTP-based communication frameworks. This specification relies on the SAML core framework
103 in [[SAMLCore](#)] and makes use of adaptations of the SAML profiles in [[SAMLBind](#)]. A separate
104 specification, [[LibertyProtSchema](#)], is used to define the Liberty protocols and messages used
105 within the profiles. Definitions for Liberty-specific terms can be found in [[LibertyGloss](#)].

106 1.1 Notation

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

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

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

117

118 Example code listings appear like this.

119

120 Note: Non-normative notes and explanations appear like this.

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

- 124 • The prefix `lib:` stands for the Liberty namespace
125 `http://projectliberty.org/schemas/core/2002/12`
- 126 • The prefix `saml:` stands for the SAML assertion namespace (see [[SAMLCore](#)]).
- 127 • The prefix `samlp:` stands for the SAML request-response protocol namespace (see
128 [[SAMLCore](#)]).
- 129 • The prefix `ds:` stands for the W3C XML signature namespace,
130 `http://www.w3.org/2000/09/xmldsig#` (see [[XMLSig](#)]).
- 131 • The prefix `SOAP-ENV:` stands for the SOAP 1.1 namespace,
132 `http://schemas.xmlsoap.org/soap/envelope` (see [[SOAP1.1](#)]).

133 Terminology from [[RFC2396](#)] is used to describe components of an HTTP URL. An HTTP URL
134 has the following form:

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

136 Sections in this document specify certain portions of the <query> component of the URL.
137 Ellipses are used to indicate additional, but unspecified, portions of the <query> component.

138 **2 Protocol Bindings**

139 The Liberty protocol bindings are defined in this section.

140 **2.1 SOAP Binding for Liberty**

141 Because the Liberty protocols are an extension of the SAML protocol (see [[SAMLCore](#)]) and a
142 SOAP protocol binding for SAML has been defined, the SOAP binding for Liberty **MUST** adhere
143 to the processing rules for the “SOAP binding for SAML” as specified in [[SAMLBind](#)] unless
144 otherwise noted. Just like SAML, the SOAP binding for Liberty uses HTTP as the transport
145 mechanism.

146 **2.2 Example of Message Exchange Using SOAP over HTTP**

147 The following is an example of the SOAP exchange for the single sign-on browser artifact profile
148 requesting an authentication assertion (the left margin whitespace added for legibility invalidates
149 the signature).

```
150 POST /authn HTTP/1.1
151 Host: idp.example.com
152 Content-type: text/xml
153 Content-length: nnnn
154 <soap-env:Envelope
155   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
156   <soap-env:Header/>
157   <soap-env:Body>
158     <samlp:Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol"
159       xmlns:lib="http://projectliberty.org/schemas/core/2002/12"
160       xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
161       xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
162       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
163       IssueInstant="2002-12-12T10:08:56Z"
164       MajorVersion="1"
165       MinorVersion="0"
166       RequestID="e4d71c43-c89a-426b-853e-a2b0c14a5ed8"
167       id="ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1"
168       xsi:type="lib:SignedSAMLRequestType">
169       <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
170         <ds:SignedInfo>
171           <ds:CanonicalizationMethod
172             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
173           </ds:CanonicalizationMethod>
174           <ds:SignatureMethod
175             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
176           </ds:SignatureMethod>
177           <ds:Reference URI="#ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1">
178             <ds:Transforms>
179               <ds:Transform
180                 Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
181               </ds:Transform>
182               <ds:Transform
183                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
184               </ds:Transform>
185             </ds:Transforms>
186             <ds:DigestMethod
187               Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
188             </ds:DigestMethod>
189             <ds:DigestValue>k6TnolGkIPKZlpUQVYok8dwkuE</ds:DigestValue>
190           </ds:Reference>
```

```
191 </ds:SignedInfo>
192 <ds:SignatureValue>
193   wXJMVoP01V1jFnWJPyOWqP5Gqm8A1+/2b5gNzF4L4LMu4yEcRtttLdPPT3bvhkwHXjL9NuOFumQ
194   5YEyiVz1NcjAxX0LfgwutvEdJb748IU4L+8obXPXfqTZLiBK1RbHCRmRvj1PIU22oGCV6EwuiWRv
195   OD60x9svtSgFJ+iXkZQ
196 </ds:SignatureValue>
197 <ds:KeyInfo>
198   <ds:X509Data>
199     <ds:X509Certificate>
200       MIIDMTCCApqgAwIBAgIBHDANBgkqhkiG9w0BAQQFADCB1TELMakGA1UEBhMCMVVMxMzA1JmVhbnVBAcT
201       AlNGMRkwFwYDQVQKEwBMAWJlcnR5IEFsbGlhbmNlMRQwEgYDVQQLLEwtJT1AgVGVzdGVycyEiMCAG
202       A1UEAxMzTGliZXJ0eSBuZXR0eS1lcnR5IENlcnR5Zmllc2EiEKMCIIGCSqGSIB3DQEJARYVcnJvZHZ3Vl
203       ekBuZW9zb2wubmV0MB4XDTAyMTIwNDU1NTg0NFoXDTEyMTIwMTE1NTg0NFowgasxMzA1JmVhbnVBAcT
204       AlVTMqswCQYDVQHEwJTRjEKMCIgA1UEChMmTGliZXJ0eSBuZXR0eS1lcnR5IENlcnR5Zmllc2EiEKMCI
205       JAYDVQQLLEw1JT1AgVGVzdGVycyBlcmllc2Nvbi1hIHNpZ25lcjEEMBUA1UEAxMOXJpY3Nzb24t
206       YS5pb3AxKDAmbGkqhkiG9w0BCQEWGjYyY2RyaWd1ZXpAZXJpY3Nzb24tYS5pb3AwZ8wDQYJKoZI
207       hvCAQEBBQADgY0AMIGJAoGBAPUoGYvJxQc5jzDnJ14TV6TaTbB3fH95ju24Z0y6HQxm6gXdJSao
208       Wh7/AIes4UcV09DC2kKS6Vow2YoXt2LIyH9HWH2tEUt1js/PUEBHEWcW3tFezM6jh5GG5rCuVPZA
209       W9eogUbfPSzOPFKUAwdHUXSDWufY1KZ93IhxOBeZgg6VAgMBAAGjeTB3MEoGCWGSAGG+EIBDQO9
210       FjtUaG1zIHNpZ25pbmVycyY2VydCB3YXNpY3JlYXRlZCBmb3IgdGVzdGluZy4gRG8gY291IHRyZDk0
211       IGl0LjA1JmVhbnVHRMEAjAAMBEGCWSAGG+EIBAQQEAwIEMDALBgNVHQ8EBAMCBsAwDQYJKoZIhvcN
212       AQEEBQADgYEAAR/HSgBpAprQwQVvWDE9pCaiduKv4/W/+hrdpXlVKSr6TIIlg4ouDCQJNos7tNuG9Z
213       AbfWtHvC551N2cfAzfs/DKqXRqcsxzL5ZUBksPpmsDoboopUv6Xm8RFsi7yB9AGaVuuQObeY/+m
214       70nOu030+F1MN3U1k2E3rOKXlU1noC0</ds:X509Certificate>
215     </ds:X509Data>
216   </ds:KeyInfo>
217 </ds:Signature>
218 <samlp:AssertionArtifact>
219   AAMluXw6+f+jyA/4XuFHqP17QDvc/LIQL9+t7YQtG1Gwk9bph0Adl+o+
220 </samlp:AssertionArtifact>
221 </samlp:Request>
222 </soap-env:Body>
223 </soap-env:Envelope>
224
```

225

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

```
228 HTTP/1.1 200 OK
229 Content-Type: text/xml
230 Content-Length: nnnn
231 <soap-env:Envelope
232   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
233   <soap-env:Header/>
234   <soap-env:Body>
235     <samlp:Response
236       xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
237       InResponseTo="RPCUk211+GVz+t11LURp51oFvJXk"
238       IssueInstant="2002-10-31T21:42:13Z" MajorVersion="1" MinorVersion="0"
239       Recipient="http://localhost:8080/sp"
240       ResponseID="LANWfL2xLybnc+BCwgY+p1/vIVAj">
241       <samlp:Status>
242         <samlp:StatusCode
243           xmlns:gns="urn:oasis:names:tc:SAML:1.0:protocol"
244           Value="gns:Success">
245         </samlp:StatusCode>
246       </samlp:Status>
247       <saml:Assertion
248         xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
249         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
250         xmlns:lib="http://projectliberty.org/schemas/core/2002/12"
251         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
252         AssertionID="SqMC8Hs2vJ7Z+t4UiLSmhKOSU00U"
253         InResponseTo="RPCUk211+GVz+t11LURp51oFvJXk"
254         IssueInstant="2002-10-31T21:42:13Z" Issuer="http://localhost:8080/idp"
255         MajorVersion="1" MinorVersion="0"
256         xsi:type="lib:AssertionType">
257         <saml:Conditions
258           NotBefore="2002-10-31T21:42:12Z"
259           NotOnOrAfter="2002-10-31T21:42:43Z">
260         <saml:AudienceRestrictionCondition>
```

```

261     <saml:Audience>http://localhost:8080/sp</saml:Audience>
262   </saml:AudienceRestrictionCondition>
263 </saml:Conditions>
264 <saml:AuthenticationStatement
265   AuthenticationInstant="2002-10-31T21:42:13Z"
266   AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
267   xsi:type="lib:AuthenticationStatementType">
268   <saml:Subject xsi:type="lib:SubjectType">
269     <saml:NameIdentifier>C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK</saml:NameIdentifier>
270     <saml:SubjectConfirmation>
271       <saml:ConfirmationMethod>
272         urn:oasis:names:tc:SAML:1.0:cm:artifact-01
273       </saml:ConfirmationMethod>
274     </saml:SubjectConfirmation>
275     <lib:IDPProvidedNameIdentifier>
276       C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
277     </lib:IDPProvidedNameIdentifier>
278   </saml:Subject>
279 </saml:AuthenticationStatement>
280 <ds:Signature>
281   <ds:SignedInfo>
282     <ds:CanonicalizationMethod
283       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
284     </ds:CanonicalizationMethod>
285     <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
286     </ds:SignatureMethod>
287     <ds:Reference URI="">
288       <ds:Transforms>
289         <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
290         </ds:Transform>
291         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
292         </ds:Transform>
293       </ds:Transforms>
294       <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
295       </ds:DigestMethod>
296       <ds:DigestValue>ZbscbqHTX9H8bBftRIWlG4Epv1A=</ds:DigestValue>
297     </ds:Reference>
298   </ds:SignedInfo>
299   <ds:SignatureValue>
300     H+q3nC3jUa1j1uKUVkcC4iTfClxeZQIFF0nvHqPS5oZhtkBaDb9qITA7gIkotaB584wXqTXwsfsu
301     IrwT5uL3r85Rj7IF6NeCeiy3K0+z3uewxyeZPz8wna449VNm0qNHYkgNak9ViNCp0/ks5MAttoPo
302     2iLOFaKu3wWG6d1G+DM=
303   </ds:SignatureValue>
304 </ds:Signature>
305 </saml:Assertion>
306 </samlp:Response>
307 </soap-env:Body>
308 </soap-env:Envelope>
309

```

3 Profiles

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

The following profile categories are defined in this document:

- **Single Sign-On and Federation:** The profiles by which a service provider obtains an authentication assertion from an identity provider facilitating single sign-on and identity federation.
- **Name Registration:** The profiles by which service providers and identity providers specify the name identifier to be used when communicating with each other about the Principal.

- 322 • **Identity Termination Notification:** The profiles by which service providers and identity
323 providers are notified of federation termination.
- 324 • **Single Logout:** The profiles by which service providers and identity providers are notified
325 of authenticated session termination.
- 326 • **Identity Provider Introduction:** The profile by which a service provider discovers which
327 identity providers a Principal may be using.

328 **3.1 Common Requirements**

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

- 331 1 All HTTP requests and responses MUST be drawn from either HTTP 1.1 (see [\[RFC2616\]](#)) or
332 HTTP 1.0 (see [\[RFC1945\]](#)). When an HTTP redirect is specified, the HTTP response MUST
333 have a status code of “302.” According to HTTP 1.1 and HTTP 1.0, the use of status code 302
334 is recommended to indicate “the requested resource resides temporarily under a different
335 URI.” The response may also include additional headers and an optional message.
- 336 2 When `https` is specified as the `<scheme>` for a URL, the HTTP connection MUST be made
337 over either SSL 3.0 (see [\[SSLv3\]](#)) or TLS 1.0 (see [\[RFC2246\]](#)) or any subsequent protocols
338 that are backwards compatible with SSL 3.0 and/or TLS 1.0. Other security protocols MAY
339 be used as long as they implement equivalent security measures.
- 340 3 Messages between providers MUST have their integrity protected, confidentiality MUST be
341 ensured and the recipient MUST authenticate the sender.
- 342 4 Providers MUST use secure transport (`https`) to achieve confidentiality and integrity
343 protection. The initiator of the secure connection MUST authenticate the server using server-
344 side X.509 certificates.
- 345 5 The authenticated identity of an identity provider MUST be securely available to a Principal
346 before the Principal presents his/her personal authentication data to that identity provider.
- 347 6 For signing and verification of protocol messages, identity and service providers SHOULD
348 use certificates and private keys that are distinct from the certificates and private keys applied
349 for SSL or TLS channel protection. Certificates and private keys MUST be suitable for long-
350 term signatures. See [\[LibertyProtSchema\]](#) for guidelines on signature verification.
- 351 7 In transactions between service providers and identity providers, requests MUST be protected
352 against replay, and received responses MUST be checked for correct correspondence with
353 issued requests. (Note: Other steps may intervene between the issuance of a request and its
354 eventual response within a multistep transaction involving redirections.) Additionally, time-
355 based assurance of freshness MAY be provided.
- 356 8 Each service provider within a circle of trust MUST be configured to enable identification of
357 the identity providers whose authentications it will accept, and each identity provider MUST
358 be configured to enable identification of the service providers it intends to serve. (Note: The
359 format of this configuration is a local matter and could, for example, be represented as lists of
360 names or as sets of X.509 certificates of other circle of trust members).
- 361 9 Circle of trust bilateral agreements on selecting certificate authorities, obtaining X.509
362 credentials, establishing and managing trusted public keys, and tracking lifecycles of
363 corresponding credentials are assumed and not in scope for this specification.

364 10 The <scheme> of the URL for SOAP endpoints MUST be `https`.

365 11 All SOAP message exchanges MUST adhere to the SOAP protocol binding for Liberty (see
366 2.1).

367 **3.1.1 User Agent**

368 A user agent, unless otherwise noted in the specific profile, MUST support the following features
369 to be interoperable with the protocols in [[LibertyProtSchema](#)] and Liberty profiles in this
370 document:

- 371 • HTTP 1.0 (see [[RFC1945](#)]) or HTTP 1.1 (see [[RFC2616](#)]).
- 372 • SSL 3.0 (see [[SSLv3](#)]) or TLS 1.0 (see [[RFC2246](#)]) or any subsequent protocols which are
373 backwards compatible with SSL 3.0 and/or TLS 1.0 either directly or via a proxy (for
374 example, a WAP gateway).
- 375 • Minimum maximum URL length of 256 bytes. See [[LibertyGloss](#)] for definition.

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

379 **3.1.2 Formatting and Encoding of Protocol Messages**

380 All Liberty protocol messages that are indicated by the profile as being communicated in the
381 <query> component of the URL MUST adhere to the formatting and encoding rules in 3.1.2.1.

382 **3.1.2.1 Encoding URL-embedded Messages**

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

386 The original Liberty XML protocol message MUST be encoded as follows:

- 387 • The <query> component parameter value MUST be the value of the Liberty XML
388 protocol message element or attribute value.
- 389 • When the original message element has multiple values, the value of the <query>
390 component parameter MUST be a space-delimited list.
- 391 • Some of the referenced protocol message elements and attributes are optional. If an
392 optional element or attribute does not appear in the original Liberty XML protocol
393 message, then the corresponding data item MUST be omitted from the URL encoded
394 message.
- 395 • URLs appearing in the URL-encoded message SHOULD NOT exceed 80 bytes in length
396 (including %-escaping overhead). Likewise, the <lib:RelayState> data value
397 SHOULD NOT exceed 80 bytes in length.
- 398 • The URL-encoding of status codes in the responses
399 `RegisterNameIdentifierResponse` and `LogoutResponse` may be taken from
400 several sources. The top level codes MUST be from SAML. Other codes (including
401 Liberty-defined values) MAY be used at the second or lower levels. The URL parameter

402 value should be interpreted as a QName with the "lib", "saml", and "samlp"
403 namespaces pre-defined to their respective namespace URIs. Query parameters with the
404 name "xmlns:prefix" can be used to map additional namespace prefixes for the purpose
405 of QName resolution, so long as the xmlns:prefix URL parameter appears before the
406 URL parameter containing the QName which needs the prefix definition.

407 As <samlp:StatusCode> elements may be nested hierarchically (see [SAMLCore]),
408 there may exist multiple values for <samlp:StatusCode> in the response messages.
409 These multiple values MUST be encoded by producing a URL-encoded space-separated
410 string as the value of this query parameter. An example is shown below:

```
411 Value=samlp%3AResponder%20lib%3AFederationDoesNotExist  
412
```

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

- 416 • Include the signature algorithm identifier as a new <query> component parameter named
417 SigAlg, but omitting the signature.
- 418 • Sign the string containing the URL-encoded message. The string to be signed MUST
419 include only the <query> part of the URL (that is, everything after ? and before
420 &Signature=). Any required URL-escaping MUST be done before signing.
- 421 • Encode the signature using base64 (see [[RFC2045](#)]).
- 422 • Add the base64-encoded signature to the encoded message as a new data item named
423 Signature.

424 Note that some characters in the base64-encoded signature value may require URL escaping
425 before insertion into the URL <query> part, as is the case for any other data item value.

426 Any items added after the Signature <query> component parameter are implicitly unsigned.

427 The service URL provided by the provider (the URL to which <query> parameters are added)
428 MUST NOT contain any pre-existing <query> parameter values.

429 The following signature algorithms (i.e., DSAwithSHA1, RSAwithSHA1) and their identifiers (the
430 URIs) MUST be supported:

- 431 • DSAwithSHA1 — <http://www.w3.org/2000/09/xmlsig#dsa-sha1>
- 432 • RSAwithSHA1 — <http://www.w3.org/2000/09/xmlsig#rsa-sha1>

433 **3.1.2.1.1 Size Limitations**

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

441 3.1.2.1.2 URL-encoded <lib:AuthnRequest>

442 The original <lib:AuthnRequest> message:

```
443 <lib:AuthnRequest
444   RequestID="[RequestID]"
445   MajorVersion="[MajorVersion]"
446   MinorVersion="[MinorVersion]"
447   IssueInstant="[IssueInstant]">
448   <lib:ProviderID>[ProviderID]</lib:ProviderID>
449   <lib:ForceAuthn>[ForceAuthn]</lib:ForceAuthn>
450   <lib:IsPassive>[IsPassive]</lib:IsPassive>
451   <lib:Federate>[Federate]</lib:Federate>
452   <lib:ProtocolProfile>[ProtocolProfile]</lib:ProtocolProfile>
453   <lib:AuthnContext>
454     <lib:AuthnContextStatementRef>[AuthnContextStatementRef]
455     </lib:AuthnContextStatementRef>
456   </lib:AuthnContext>
457   <lib:RelayState>[RelayState]</lib:RelayState>
458   <lib:AuthnContextComparison>[AuthnContextComparison]
459     </lib:AuthnContextComparison>
460 </lib:AuthnRequest>
```

461

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

464 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,
465 ForceAuthn, IsPassive, Federate, ProtocolProfile,
466 AuthnContextStatementRef, AuthnContextClassRef,
467 AuthnContextComparison, RelayState.

- 468 • Maximum size: 748 bytes + 81 * number of AuthnContextClassRef or
469 AuthnContextStatementRefs

- 470 • Example of <lib:AuthnRequest> message URL-encoded and signed (772 bytes):

```
471 http://idp.example.com/authn?RequestID=RMvY34pg%2FV9aGJ5yw0HL0AcjCqQF&MajorVersion=1&MinorVersi
472 on=0&IssueInstant=2002-05
473 15T00%3A58%3A19&ProviderID=http%3A%2F%2Fsp.example.com%2Fliberty%2F&ForceAuthn=true&IsPassive=f
474 alse&Federate=true&ProtocolProfile=http%3A%2F%2Fprojectliberty.org%2Fprofiles%2Fbrws-
475 post&AuthnContextClassRef=http%3A%2F%2Fprojectliberty.org%2Fauthnctx%2Fprofiles%2Fpassword-
476 over-
477 HTTP&RelayState=03mhakSms5tMQ0WRDCEzpF7BNcywZa75FwIcSSEPvbkofXaQHCuNnc5yChIdDlWc7JBV9Xbw3avRBK7
478 VFsPl2X&SigAlg=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23rsa-
479 sha1&Signature=EoD8bNr2jEoe%2Fumon6oU%2FZGIIF7gbJAe4MLUUMrD%2BPP7P8Yf3gfdZG2qPJdNAJkzVHGfO8W8Dz
480 pQ%0D%0AsDTTd5VP9MLPcvxbFQoF0CJjmvL26cPsuc54q7ourcH0jJ%2F2UkDq4DA1YlZ5kPIg%2BtrykgLz0U%2BS%0D%0
481 ANqpNHkjH6W3YkGv7Rbs%3D
```

482 3.1.2.1.3 URL-Encoded <lib:FederationTerminationNotification>

483 The original <lib:FederationTerminationNotification> message:

```
484 <lib:FederationTerminationNotification ...
485   RequestID="[RequestID]"
486   MajorVersion="[MajorVersion]"
487   MinorVersion="[MinorVersion]"
488   IssueInstant="[IssueInstant]">
489     <lib:ProviderID>[ProviderID]</lib:ProviderID>
490     <saml:NameIdentifier
491       NameQualifier="[NameQualifier]"
492       Format="[NameFormat]">[NameIdentifier]</saml:NameIdentifier>
493     <lib:RelayState>[RelayState]</lib:RelayState>
494 </lib:FederationTerminationNotification>
```

495

- 496 • Data elements that **MUST** be included in the encoded data with their values as indicated in
497 brackets above if present in the original message:
- 498 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,
499 NameQualifier, NameFormat, NameIdentifier, RelayState.

500 **3.1.2.1.4 URL-Encoded <lib:LogoutRequest>**

501 The original <lib:LogoutRequest> message:

```
502 <lib:LogoutRequest ...  
503   RequestID="[RequestID]"  
504   MajorVersion="[MajorVersion]"  
505   MinorVersion="[MinorVersion]"  
506   IssueInstant="[IssueInstant]">  
507     <lib:ProviderID>[ProviderID]</lib:ProviderID>  
508     <saml:NameIdentifier  
509       NameQualifier="[NameQualifier]"  
510       Format="[NameFormat]">  
511       [NameIdentifier]  
512     </saml:NameIdentifier>  
513     <lib:SessionIndex>[SessionIndex]</lib:SessionIndex>  
514     <lib:RelayState>[RelayState]</lib:RelayState>  
515 </lib:LogoutRequest>
```

- 516
- 517 • Data elements that **MUST** be included in the encoded data with their values as indicated in
518 brackets above if present in the original message:
- 519 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,
520 NameQualifier, NameFormat, NameIdentifier, SessionIndex,
521 RelayState.

522 **3.1.2.1.5 URL-Encoded <lib:LogoutResponse>**

523 The <lib:LogoutResponse> response message:

```
524 <lib:LogoutResponse  
525   ResponseID="[ResponseID]"  
526   InResponseTo="[InResponseTo]"  
527   MajorVersion="[MajorVersion]"  
528   MinorVersion="[MinorVersion]"  
529   IssueInstant="[IssueInstant]"  
530   Recipient="[Recipient]">  
531     <lib:ProviderID>[ProviderID]</lib:ProviderID>  
532     <samlp:Status>  
533       <samlp:StatusCode Value="[Value]" />  
534     </samlp:Status>  
535     <lib:RelayState>[RelayState]</lib:RelayState>  
536 </lib:LogoutResponse>
```

- 537
- 538 • Data elements that **MUST** be included in the encoded data with their values as indicated in
539 brackets above if present in the original message:
- 540 ResponseID, InResponseTo, MajorVersion, MinorVersion,
541 IssueInstant, Recipient, ProviderID, Value, RelayState.
- 542 • The <lib:LogoutResponse> message may contain nested status code information.
543 Multiple values **MUST** be URL-encoded by creating a space-separated list (see general
544 requirements at top of section 3.1.2.1.5).

545 3.1.2.1.6 URL-Encoded <lib:RegisterNameIdentifierRequest>

546 The original <lib:RegisterNameIdentifierRequest> message:

```
547 <lib:RegisterNameIdentifierRequest
548   RequestID="[RequestID]"
549   MajorVersion="[MajorVersion]"
550   MinorVersion="[MinorVersion]"
551   IssueInstant="[IssueInstant]">
552   <lib:ProviderID>[ProviderID]</lib:ProviderID>
553   <lib:IDPProvidedNameIdentifier
554     NameQualifier="[IDPNameQualifier]"
555     Format="[IDPNameFormat]">[IDPProvidedNameIdentifier]
556   </lib:IDPProvidedNameIdentifier>
557   <lib:SPPProvidedNameIdentifier
558     NameQualifier="[SPNameQualifier]"
559     Format="[SPNameFormat]">[SPPProvidedNameIdentifier]
560   </lib:SPPProvidedNameIdentifier>
561   <lib:OldProvidedNameIdentifier
562     NameQualifier="[OldNameQualifier]"
563     Format="[OldNameFormat]">[OldProvidedNameIdentifier]
564   </lib:OldProvidedNameIdentifier>
565   <lib:RelayState>[RelayState]</lib:RelayState>
566 </lib:RegisterNameIdentifierRequest>
```

567

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

570 RequestID, MajorVersion, MinorVersion, IssueInstant, ProviderID,
571 IDPNameQualifier, IDPNameFormat, IDPProvidedNameIdentifier,
572 SPNameQualifier, SPNameFormat, SPPProvidedNameIdentifier,
573 OldNameQualifier, OldNameFormat, OldProvidedNameIdentifier,
574 RelayState

575 3.1.2.1.7 URL-Encoded <lib:RegisterNameIdentifierResponse>

576 The <lib:RegisterNameIdentifierResponse> response message:

```
577 <lib:RegisterNameIdentifierResponse
578   ResponseID="[ResponseID]"
579   InResponseTo="[InResponseTo]"
580   MajorVersion="[MajorVersion]"
581   MinorVersion="[MinorVersion]"
582   IssueInstant="[IssueInstant]"
583   Recipient="[Recipient]">
584   <lib:ProviderID>[ProviderID]</lib:ProviderID>
585   <samlp:Status>
586     <samlp:StatusCode Value="[Value]" />
587   </samlp:Status>
588   <lib:RelayState>[RelayState]</lib:RelayState>
589 </lib:RegisterNameIdentifierResponse>
```

590

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

593 ResponseID, InResponseTo, MajorVersion, MinorVersion,
594 IssueInstant, Recipient, ProviderID, Value, RelayState

- 595 • The <lib:RegisterNameIdentifierResponse> message may contain nested status
596 code information. Multiple values MUST be URL-encoded by creating a space-separated
597 list (see general requirements at top of section 3.1.2.1).

598 **3.1.3 Provider Metadata**

599 The majority of the Liberty profiles defined in this document rely on metadata that specify the
600 policies that govern the behavior of the service provider or identity provider. These provider
601 metadata are typically shared out of band between an identity provider and a service provider prior
602 to the exchange of Liberty protocol messages. The provider metadata relevant to each profile are
603 listed in this document at the beginning of the profile category. Refer to [[LibertyProtSchema](#)] for a
604 complete enumeration of the Liberty provider metadata elements and their associated schema.

605 **3.2 Single Sign-On and Federation Profiles**

606 This section defines the profiles by which a service provider obtains an authentication assertion
607 from an identity provider to facilitate single sign-on. Additionally, the single sign-on profiles can
608 be used as a means of federating an identity from a service provider to an identity provider through
609 the use of the <Federate> element in the <lib:AuthnRequest> protocol message as specified
610 in [[LibertyProtSchema](#)].

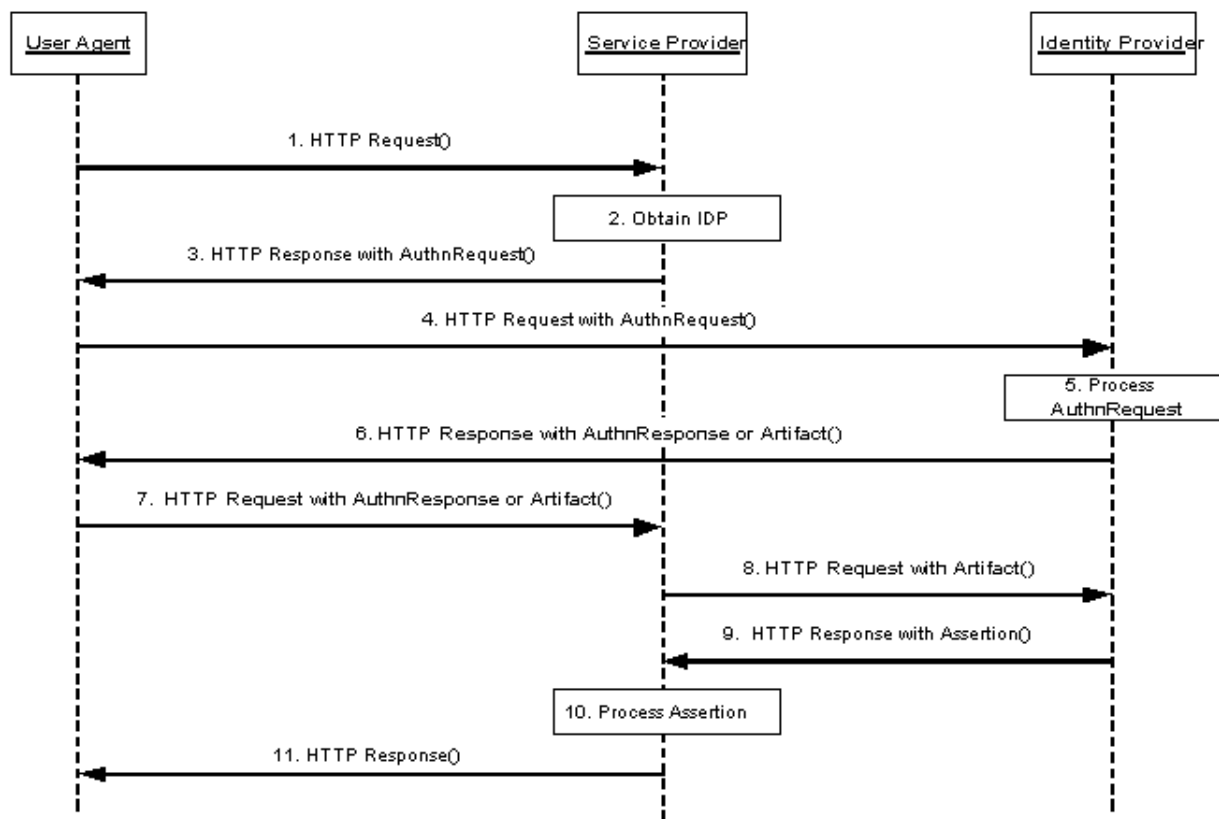
611 The single sign-on profiles make use of the following metadata elements, as defined in
612 [[LibertyProtSchema](#)].

- 613 • `ProviderID` — Used to uniquely identify the service provider to the identity provider
614 and is documented in these profiles as “service provider ID.”
- 615 • `SingleSignOnServiceURL` — The URL at the identity provider that the service
616 provider should use when sending single sign-on and federation requests. It is documented
617 in these profiles as “single sign-on service URL.”
- 618 • `AssertionConsumerServiceURL` — The URL at the service provider that an identity
619 provider should use when sending single sign-on or federation responses. It is documented
620 in these profiles as “assertion consumer service URL.”
- 621 • `SOAPEndpoint` — The SOAP endpoint location at the service provider or identity
622 provider to which Liberty SOAP messages are sent.

623 **3.2.1 Common Interactions and Processing Rules**

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

626 All single sign-on profiles can be described by one interaction diagram, provided that different
627 messages are optional in different profiles and that the actual content of the messages may differ
628 slightly. Where interactions and messages differ or are optional, they are called out and detailed
629 within the specific single sign-on profiles. Figure 1 represents the basic template of interactions for
630 achieving single sign-on and should be used as the baseline for all single sign-on profiles.



631

632

Figure 1: Basic single sign-on profile

633 **Step 1: HTTP Request**

634 In step 1, the user agent accesses the intersite transfer service at the service provider with
 635 information about the desired target attached to the URL. Typically, access to the intersite transfer
 636 service occurs via a redirection by the service provider in response to a user agent request for a
 637 restricted resource.

638 It is RECOMMENDED that the HTTP Request URI contain a <query> component at its end
 639 where

640 <query>=...RelayState=<return URL>...

641 The <query> component can be used to convey information about the originally requested
 642 resource at the service provider. It is RECOMMENDED that the <query> parameter be
 643 named RelayState and its value be the URL originally requested by the user agent.

644
 645 It is RECOMMENDED that the HTTP request be made over either SSL 3.0 (see [SSLv3]) or TLS
 646 1.0 (see [RFC2246]) to maintain confidentiality and message integrity in step 1.

647 **Step 2: Obtain Identity Provider**

648 In step 2, the service provider obtains the address of the appropriate identity provider to redirect
649 the user agent to in step 3. The means by which the identity provider address is obtained is
650 implementation-dependent and up to the service provider. The service provider MAY use the
651 Liberty identity provider introduction profile in this step.

652 **Step 3: HTTP Response with <AuthnRequest>**

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

655 The form and contents of the HTTP response in this step are profile-dependent.

656 **Step 4: HTTP Request with <AuthnRequest>**

657 In step 4, the user agent accesses the identity provider's single sign-on service URL with the
658 <lib:AuthnRequest> information.

659 **Step 5: Processing <AuthnRequest>**

660 In step 5, the identity provider MUST process the <lib:AuthnRequest> message according to
661 the rules specified in [[LibertyProtSchema](#)].

662 If the Principal has not yet been authenticated with the identity provider, authentication at the
663 identity provider MAY occur in this step. The identity provider MAY obtain consent from the
664 Principal for federation, or otherwise consult the Principal. To this end the identify provider MAY
665 return to the HTTP request any HTTP response; including but not limited to HTTP Authentication,
666 HTTP redirect, or content. The identity provider SHOULD respect the HTTP User-Agent and
667 Accept headers and SHOULD avoid responding with content-types that the User-Agent may not
668 be able to accept. Authentication of the Principal by the identity provider is dependent upon the
669 <lib:AuthnRequest> message content.

670 In case the identity provider responds to the user agent with a form, it is RECOMMENDED that
671 the <input> parameters of the form be named according to [[RFC3106](#)] whenever possible.

672 **Step 6: HTTP Response with <AuthnResponse> or Artifact**

673 In step 6, the identity provider MUST respond to the user agent with a <lib:AuthnResponse>,
674 a SAML artifact, or an error.

675 The form and contents of the HTTP response in this step are profile-dependent.

676 **Step 7: HTTP Request with <AuthnResponse> or Artifact**

677 In step 7, the user agent accesses the assertion consumer service URL at the service provider with
678 a <lib:AuthnResponse> or a SAML artifact.

679 The form and contents of the HTTP request in this step are profile-dependent.

680 **Step 8: HTTP Request with Artifact**

681 Step 8 is required only for single sign-on profiles that use a SAML artifact.

682 In this step the service provider, in effect, dereferences the single SAML artifact in its possession
683 to acquire the authentication assertion that corresponds to the artifact.

684 The service provider **MUST** send a `<samlp:Request>` SOAP message to the identity provider's
685 SOAP endpoint, requesting the assertion by supplying the SAML assertion artifact in the
686 `<samlp:AssertionArtifact>` element as specified in [[SAMLBind](#)].

687 The `<samlp:Request>` **MUST** be digitally signed by the service provider. The
688 `<samlp:Request>` **MUST** be signed in accordance with Liberty signing guidelines (Sections
689 3.1.3 and 3.1.5 in [[LibertyProtSchema](#)]). An `id` attribute may be added to the `<samlp:Request>`
690 by declaring an alternate type, `<lib:SignedSAMLRequestType>`. This `id` attribute may be
691 used for signing. The use of the alternate type is as follows:

```
692 <soap-env:Envelope  
693   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">  
694   <soap-env:Header/>  
695   <soap-env:Body>  
696     <samlp:Request  
697       xsi:type="lib:SignedSAMLRequestType"  
698       id="x"  
699       xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"  
700       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
701       IssueInstant="2002-10-31T21:42:14Z"  
702       MajorVersion="1"  
703       MinorVersion="0"  
704       RequestID="2H+PRhYSFYXozOD6r6PZ4YqyKfft">  
705       ...  
706     </samlp:Request>  
707   </soap-env:Body>  
708 </soap-env:Envelope>
```

710 **Step 9: HTTP Response with Assertion**

711 Step 9 is required only for single sign-on profiles that use a SAML artifact.

712 In this step if the identity provider is able to find or construct the requested assertion, it responds
713 with a `<samlp:Response>` SOAP message with the requested `<saml:Assertion>`. Otherwise,
714 it returns an appropriate status code, as defined within the “SOAP binding for SAML” (see
715 [[SAMLBind](#)]) and the [[LibertyProtSchema](#)].

716 The `<samlp:Response>` message **MAY** be digitally signed. The `<saml:Assertion>` contained
717 in the message **MUST** be digitally signed by the identity provider.

718 The `<saml:Assertion>` elements contained within the `<samlp:Response>` message returned
719 by the identity provider **MUST** include a `<saml:NameIdentifier>` element. If the service
720 provider has registered a name identifier, *i.e.*, the `SPProvidedNameIdentifier`, that value will
721 be used. If the service provider has not registered a name identifier, the name identifier provided
722 by the identity provider will be used. When the identity provider returns multiple assertions within
723 `<samlp:Response>`, it **MUST** return exactly one `<saml:Assertion>` for each SAML artifact
724 found in the corresponding `<samlp:Request>` element. The case where fewer or greater number
725 of assertions is returned within the `<samlp:Response>` element **MUST** be treated as an error
726 state by the service provider. The identity provider **MUST** return a response with zero assertions if

727 a `<samlp:Request>` is received from any service provider other than the service provider for
728 which the SAML artifact was originally issued.

729 The `<saml:ConfirmationMethod>` element of the assertion **MUST** be set to the value specified
730 in [[SAMLCore](#)] for “SAML Artifact,” and the `<saml:SubjectConfirmationData>` element
731 **MUST** be present with its value being the SAML artifact supplied to obtain the assertion.

732 **Step 10: Process Assertion**

733 In step 10, the service provider processes the `<saml:Assertion>` returned in the
734 `<samlp:Response>` or `<lib:AuthnResponse>` protocol message to determine its validity and
735 how to respond to the Principal’s original request. The signature on the `<saml:Assertion>` must
736 be verified.

737 The service provider processing of the assertion **MUST** adhere to the rules defined in [[SAMLCore](#)]
738 for things such as assertion `<saml:Conditions>` and `<saml:Advice>`.

739 The service provider **MAY** obtain authentication context information for the Principal’s current
740 session from the `<lib:AuthnContext>` element contained in the `<saml:advice>`. Similarly,
741 the information in the `<lib:RelayState>` element **MAY** be obtained and used in further
742 processing by the service provider.

743 **Step 11: HTTP Response**

744 In step 11, the user agent is sent an HTTP response that either allows or denies access to the
745 originally requested resource.

746 **3.2.2 Liberty Browser Artifact Profile**

747 The Liberty browser artifact profile relies on a reference to the needed assertion traveling in a
748 SAML artifact, which the service provider must dereference from the identity provider to
749 determine whether the Principal is authenticated. This profile is an adaptation of the
750 “Browser/artifact profile” for SAML as documented in [[SAMLBind](#)]. See Figure 2.

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

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

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

757 **3.2.2.1 Interactions**

758 Figure 2 illustrates the Liberty browser artifact profile for single sign-on.

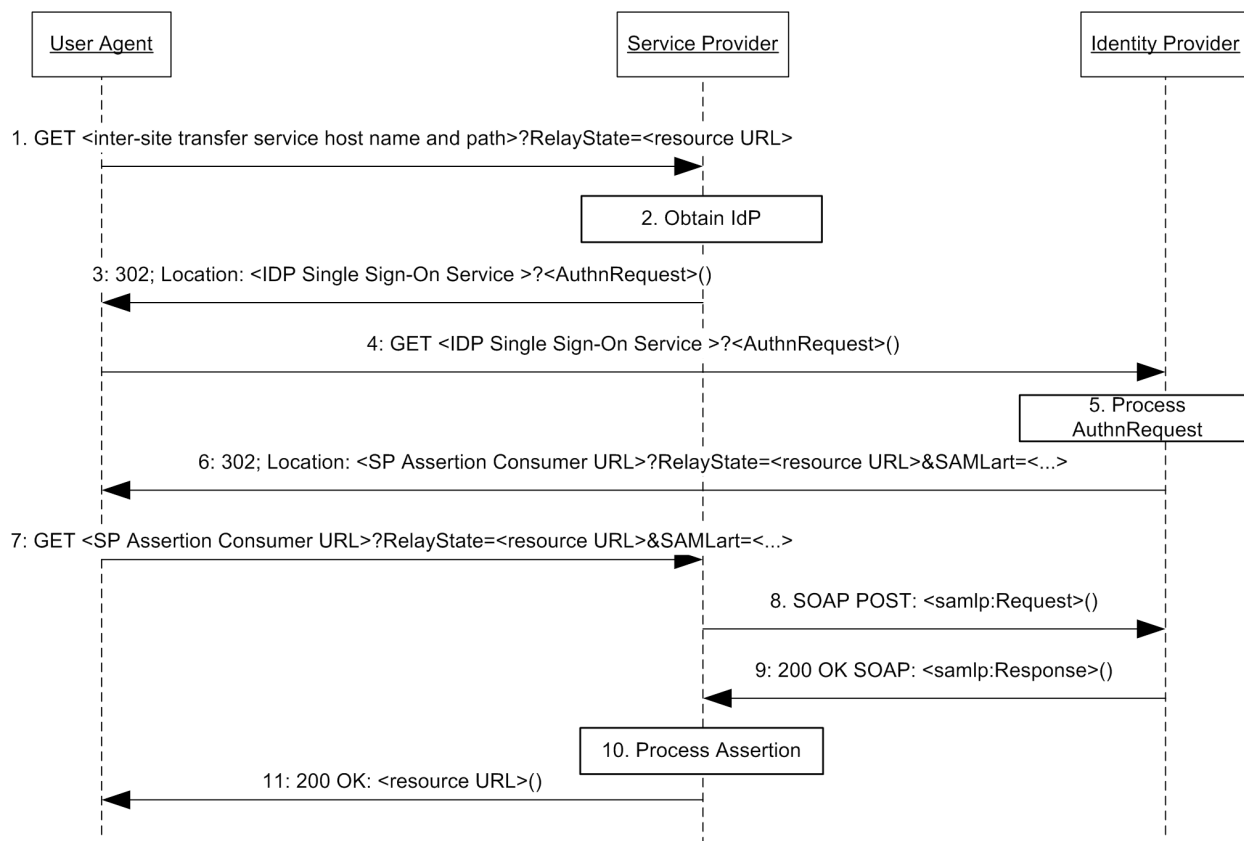


Figure 2: Liberty browser artifact profile for single sign-on

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

When implementing this profile, all processing rules defined in 3.2.1 for the single sign-on profiles MUST be followed. Additionally, the following rules MUST be observed as they relate to steps 3, 6 and 7:

Step 3: Single sign on Service with <AuthnRequest>

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

The redirection MUST adhere to the following rules:

- The Location HTTP header MUST be set to the identity provider's single sign-on service URL.
- The identity provider's single sign-on service URL MUST specify `https` as the URL scheme; if another scheme is specified, the service provider MUST NOT redirect to the identity provider.

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

- 778 • The Location HTTP header MUST include a `<query>` component containing the
779 `<lib:AuthnRequest>` protocol message as defined in [[LibertyProtSchema](#)] with
780 formatting as specified in 3.1.2.

781 Note: The `<lib:RelayState>` element of the `<lib:AuthnRequest>` message
782 can be used by the service provider to help maintain state information during the
783 single sign-on and federation process. For example, the originally requested
784 resource (that is, `RelayState` in step 1) could be stored as the value for the
785 `<lib:RelayState>` element, which would then be returned to the service
786 provider in the `<lib:AuthnResponse>` in step 7. The service provider could
787 then use this information to know how to formulate the HTTP response to the user
788 agent in step 11.

789 The HTTP response MUST take the following form:

```
790 <HTTP-Version> 302 <Reason Phrase>  
791 <other headers>  
792 Location: https://<Identity Provider Single Sign-On Service host name and path>?<query>  
793 <other HTTP 1.0 or 1.1 components>
```

794 where

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

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

798 `<query>=` ...`<URL-encoded AuthnRequest>` ...

799 A `<query>` component MUST contain a single authentication request.

800 Step 6: Redirecting to the Service Provider

801 In step 6, the identity provider performs a redirection to the service provider's assertion consumer
802 service URL including a SAML artifact in the `<query>` component of the URL.

803 The redirection MUST adhere to the following rules:

- 804 • The Location HTTP header MUST be set to the service provider's assertion consumer
805 service URL, the value of which was determined based upon the `<lib:ProviderID>`
806 element of the `<lib:AuthnRequest>` message.
- 807 • The service provider's assertion consumer service URL MUST specify `https` as the URL
808 scheme; if another scheme is specified, the identity provider MUST NOT redirect to the
809 service provider.
- 810 • The Location HTTP header MUST include a `<query>` parameter `SAMLart`, the value of
811 which is the SAML artifact on success or on failure. In the case of failure, the status
812 information will be communicated in the `<samlp:Response>` returned in step 9.
813 Additionally, if the `<lib:AuthnRequest>` processed in step 5 included a value for the
814 `<lib:RelayState>` element, then a parameter named `RelayState` with a value set to
815 the value of the `<lib:RelayState>` element MUST be included in the `<query>`
816 component.

817 The HTTP response MUST take the following form:

```
818 <HTTP-Version> 302 <Reason Phrase>  
819 <other headers>  
820 Location: https://<Service Provider assertion consumer service URL>?<query>
```

821 <other HTTP 1.0 or 1.1 components>

822 where

823 <Service Provider assertion consumer service URL>

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

826 <query>= ...SAMLart=<SAML Artifact>...RelayState=<resource URL> ...

827 At least one SAML artifact MUST be included in the <query> component. A single
828 RelayState MUST be included if a value for <RelayState> was provided in the
829 <lib:AuthnRequest>. If more than one SAML artifact is included the <query>
830 component, all artifacts MUST have the same IdentityProviderID.

831 Step 7: Accessing the Assertion Consumer Service

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

834 3.2.2.2 Artifact Format

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

836 SAML_artifact := B64 (TypeCode RemainingArtifact)
837 TypeCode := Byte1Byte2

838

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

843 TypeCode := 0x0003
844 RemainingArtifact := IdentityProviderSuccinctID AssertionHandle
845 IdentityProviderSuccinctID:= 20-byte_sequence
846 AssertionHandle := 20-byte_sequence

847

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

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

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

- 861 • Each identity provider selects a single identification URL, corresponding to the provider
862 metadata element ProviderID specified in [[LibertyProtSchema](#)].

- 863 • The identity provider constructs the IdentityProviderSuccinctID component of the artifact by
864 taking the SHA-1 hash of the identification URL as a 20-byte binary value. Note that the
865 IdentityProviderSuccinctID value, used to construct the artifact, is not encoded in
866 hexadecimal. The AssertionHandle value is constructed from a cryptographically
867 strong random or pseudo-random number sequence (see [RFC1750]) generated by the
868 identity provider. The sequence consists of values of at least eight bytes in size. These
869 values should be padded to a total length of 20 bytes.

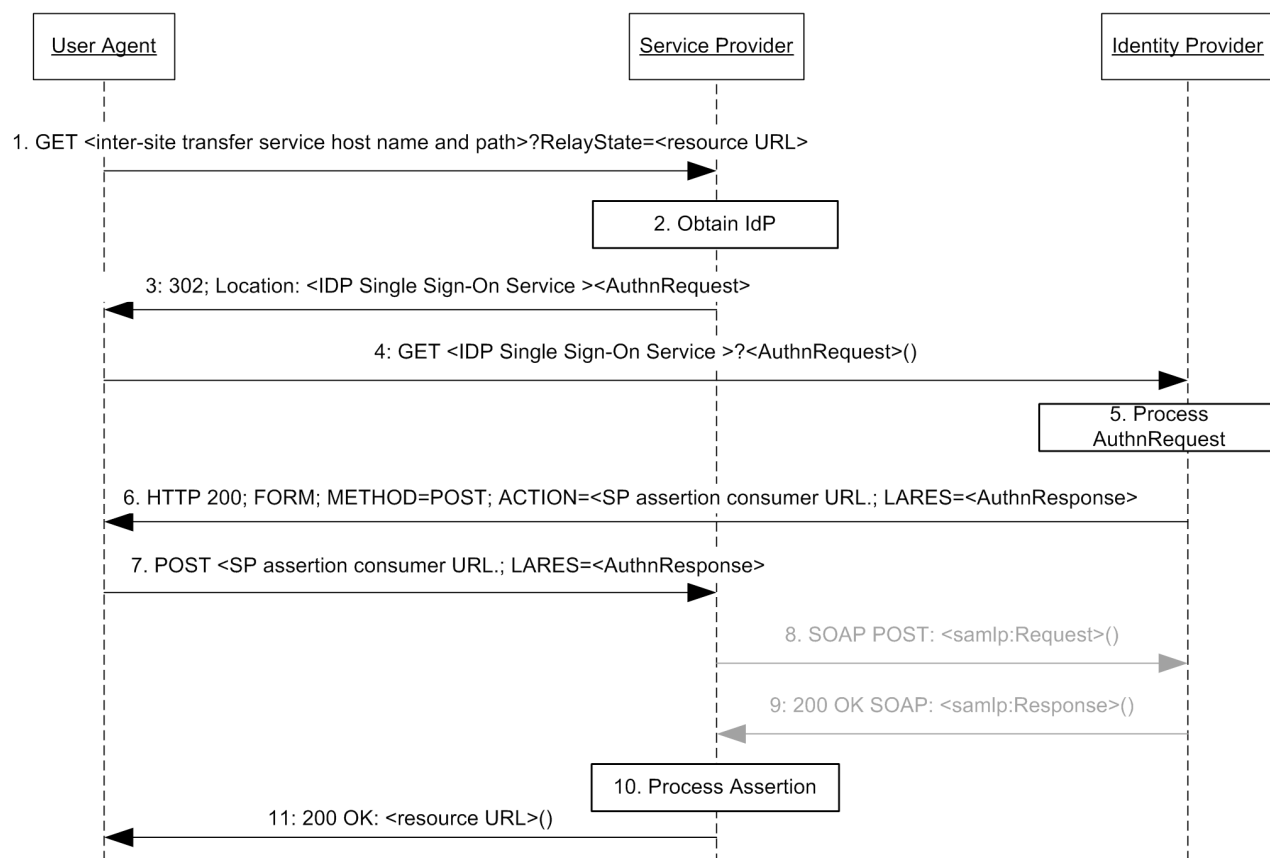
3.2.3 Liberty Browser POST Profile

871 The Liberty browser POST profile allows authentication information to be supplied to an identity
872 provider without the use of an artifact. Figure 3 diagrams the interactions between parties in the
873 Liberty POST profile. This profile is an adaptation of the “Browser/post profile” for SAML as
874 documented in [[SAMLBind](#)].

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

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

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



880
881 **Figure 3: Liberty browser POST profile for single sign-on**

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

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

887 **Step 3: Single Sign On Service with <AuthnRequest>:**

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

890 The redirection MUST adhere to the following rules:

- 891 • The Location HTTP header MUST be set to the identity provider's single sign-on service
892 URL.
- 893 • The identity provider's single sign-on service URL MUST specify `https` as the URL
894 scheme; if another scheme is specified, the service provider MUST NOT redirect to the
895 identity provider.

896 Note: Future protocols may be adopted and enabled to work within this
897 framework. Therefore, implementers are encouraged to not hardcode a reliance on
898 `https`.

- 899 • The Location HTTP header MUST include a `<query>` component containing the
900 `<lib:AuthnRequest>` protocol message as defined in [[LibertyProtSchema](#)] with
901 formatting as specified in 3.1.2.

902 Note: The `<lib:RelayState>` element of the `<lib:AuthnRequest>` message
903 can be used by the service provider to help maintain state information during the
904 single sign-on and federation process. For example, the originally requested
905 resource (that is, `RelayState` in step 1) could be stored as the value for the
906 `<lib:RelayState>` element, which would then be returned to the service
907 provider in the `<lib:AuthnResponse>` in step 7. The service provider could then
908 use this information to know how to formulate the HTTP response to the user agent
909 in step 11.

910 The HTTP response MUST take the following form:

```
911 <HTTP-Version> 302 <Reason Phrase>  
912 <other headers>  
913 Location: https://<Identity Provider Single Sign-On Service host name and path>?<query>  
914 <other HTTP 1.0 or 1.1 components>
```

915 where

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

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

919 `<query>= ...<URL-encoded AuthnRequest> ...`

920 A `<query>` component MUST contain a single authentication request.

921 **Step 6: Generating and Supplying the <AuthnResponse>**

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

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

931 Multiple <saml:Assertion> elements MAY be included in the response. The identity provider
932 MUST digitally sign all the assertions included in the response.

933 The <saml:ConfirmationMethod> element of the assertion MUST be set to the value
934 specified in [[SAMLCore](#)] for "Assertion Bearer."

935 **Step 7: Posting the Form Containing the <AuthnResponse>**

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

938 **3.2.4 Liberty WML POST Profile**

939 The Liberty WML POST profile relies on the use of WML events to instruct a WML browser to
940 submit a HTTP form. This profile is an adaptation of the "Browser/form post profile" for SAML
941 as documented in [[SAMLBind](#)]. See Figure 4

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

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

945 WML browsers are typical on mobile handsets. The browsers on such handsets communicate via a
946 dedicated proxy, a WAP gateway. This proxy converts the Wireless Session Protocol of the
947 handset into HTTP. Note: The service provider and identity provider will be contacted using only
948 HTTP.

949 The WML profile described in this section allows for the transportation of signed Liberty
950 messages that are up to approximately 1100 bytes; the length is limited by the overall size of the
951 WML deck. Many WAP browsers do not accept WML decks that are larger than 1300 bytes (after
952 WML tokenizing).

953 A user agent for this profile, typically a standard WAP browser on a mobile handset, MUST
954 support WAP WML 1.0, 1.1, 1.2, or 1.3 (see [[WML1.3](#)]) in addition to the features listed in 3.1.

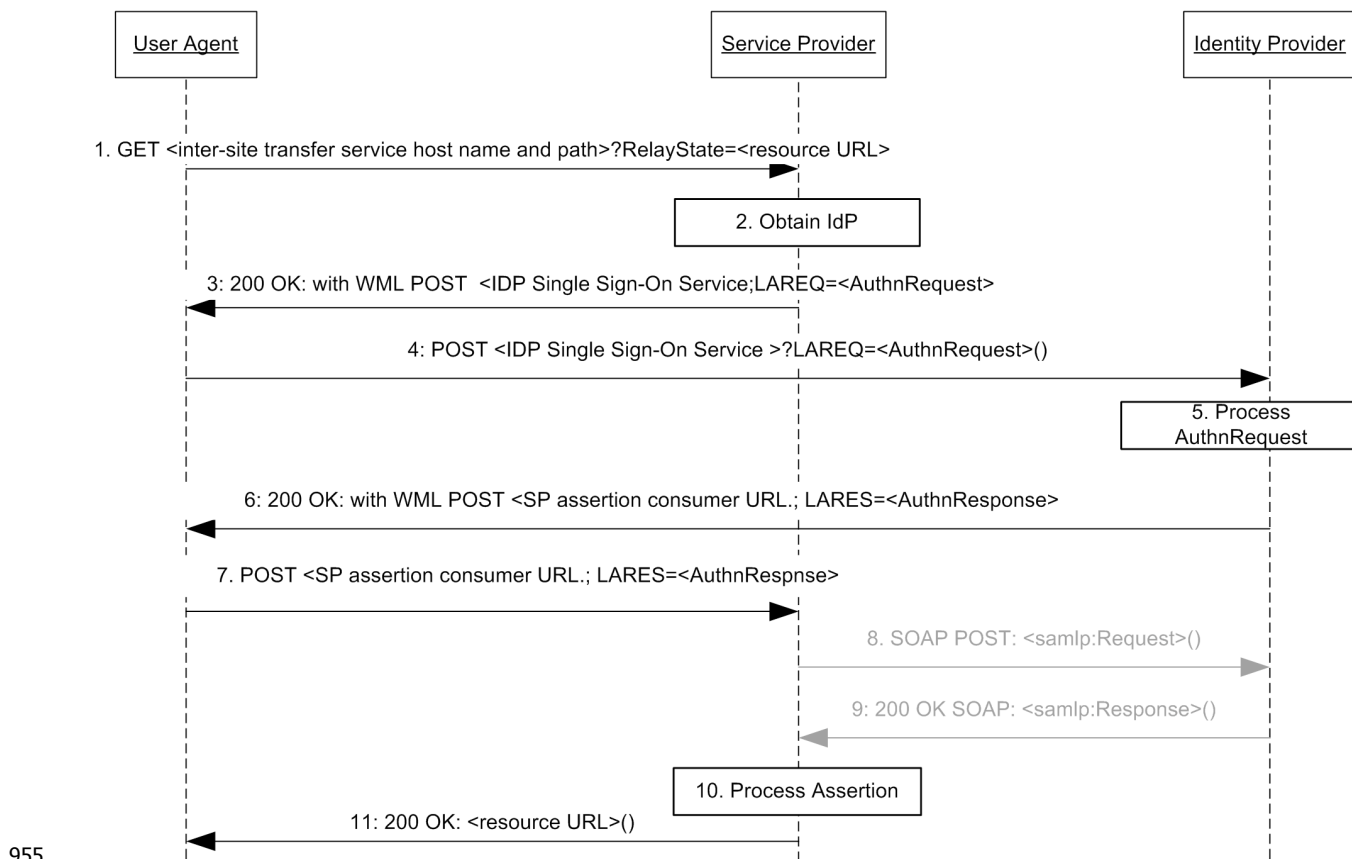


Figure 4: Liberty WML POST profile for single sign-on

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

959 When implementing this profile, all processing rules defined in 3.2.1 for single sign-on profiles
960 MUST be followed with the exception that steps 8 and 9 MUST be omitted. Additionally, the
961 following rules MUST be observed as they relate to steps 3, 4, 6, and 7:

Step 3: HTTP Response with <AuthnRequest>

963 In step 3, the service provider’s intersite transfer service responds and instructs the user agent to
964 POST an <lib:AuthnRequest> to the single sign-on service URL at the identity provider.

965 The form contents MUST contain the field LAREQ with the value of the <lib:AuthnRequest>
966 protocol message as defined in [LibertyProtSchema]. The <lib:AuthnRequest> MUST be
967 encoded by applying a base64 transformation (refer to [RFC2045]) to the <lib:AuthnRequest>
968 and all its elements. The identity provider’s single sign-on service URL used as the target of the
969 form POST MUST specify https as the URL scheme; if another scheme is specified, the service
970 provider MUST NOT issue the POST of the <lib:AuthnRequest> to the identity provider.

971 Note: One method for seamlessly instructing the user agent to POST the
972 <lib:AuthnRequest> is to include a WML deck (see Chapter 17 in [HTML4])
973 within the HTTP 200 response. The following is an example of how the WML
974 code could be structured:

```
975 ...
976 <wml>
977 <card id="redirect" title="Log In">
978   <onenterforward>
979     <go method="post" href="<Identity Provider Single Sign-On service URL>" >
980     <postfield name="LAREQ" Value="(<lib:AuthnRequest>)" />
981   </go>
982   </onenterforward>
983   <onenterbackward>
984     <prev/>
985   </onenterbackward>
986   <p>
987     Contacting IdP. Please wait...
988   </p>
989 ...
990 </card>
991 ...
992 </wml>
```

993

994 It is recommended that the `<go>` element be contained within a
995 `<onenterforward>` element of the first `<card>` in the WML deck. The `<go>`
996 element will ensure that the browser will post the authentication request as soon as
997 the WML code is processed. In addition it is recommended to add an
998 `<onenterbackward>` element to ensure that a Principal will not be presented
999 with the redirect card when navigating backwards.

1000 **Step 4: HTTP Request with `<AuthnRequest>`**

1001 In step 4, the user agent issues the HTTP POST request containing the `<lib:AuthnRequest>` to
1002 the identity provider.

1003 **Step 6: HTTP Response with `<AuthnResponse>`**

1004 In step 6, the identity provider's single sign-on service instructs the user agent to POST a
1005 `<lib:AuthnResponse>` to the assertion consumer service URL at the service provider.

1006 The form MUST be constructed so that it requests a POST to the service provider's assertion
1007 consumer service URL with the form contents that contain the field `LARES` with the value being
1008 the `<lib:AuthnResponse>` protocol message as defined in [[LibertyProtSchema](#)]. The
1009 `<lib:AuthnResponse>` MUST be encoded by applying a base64 transformation (refer to
1010 [[RFC2045](#)]) to the `<lib:AuthnResponse>` and all its elements. Multiple SAML assertions
1011 MAY be included in the response. The identity provider MUST digitally sign the assertions
1012 included in the response. The service provider's assertion consumer service URL used as the target
1013 of the form POST MUST specify `https` as the URL scheme; if another scheme is specified, it
1014 MUST be treated as an error by the identity provider.

1015 The `<saml:ConfirmationMethod>` element of the assertion MUST be set to the value specified
1016 in [[SAMLCore](#)] for "Assertion Bearer."

1017 Note: As in step 3, one way of achieving this step is to use a WML deck.

1018 **Step 7: HTTP POST with `<AuthnResponse>`**

1019 In step 7, the user agent issues the HTTP POST request containing the `<lib:AuthnResponse>`
1020 to the service provider.

1021 3.2.5 Liberty-Enabled Client and Proxy Profile

1022 The Liberty-enabled client and proxy profile specifies interactions between Liberty-enabled clients
1023 and/or proxies, service providers, and identity providers. See Figure 5. A Liberty-enabled client is
1024 a client that has, or knows how to obtain, knowledge about the identity provider that the Principal
1025 wishes to use with the service provider. In addition a Liberty-enabled client receives and sends
1026 Liberty messages in the body of HTTP requests and responses. Therefore, Liberty-enabled clients
1027 have no restrictions on the size of the Liberty protocol messages.

1028 A Liberty-enabled proxy is a HTTP proxy (typically a WAP gateway) that emulates a Liberty-
1029 enabled client. Unless stated otherwise, all statements referring to LECP are to be understood as
1030 statements about both Liberty-enabled clients as well as Liberty-enabled proxies.

1031 The following URI-based identifier must be used when referencing this specific profile (for
1032 example, <lib:ProtocolProfile> element of the <lib:AuthnRequest> message)

1033 URI: `http://projectliberty.org/profiles/lecp`

1034 All LECPs, in addition to meeting the common requirements for profiles in 3.1, MUST indicate
1035 that it is a LECP by including a Liberty-Enabled header or entry in the value of the HTTP User-
1036 Agent header for each HTTP request they make. The preferred method is the Liberty-Enabled
1037 header. The formats of the Liberty-Enabled header and User-Agent header entry are defined
1038 3.2.5.1.

1039 3.2.5.1 Liberty-Enabled Indications

1040 A LECP SHOULD add the Liberty-Enabled header to each HTTP request. The Liberty-Enabled
1041 header MUST be named `Liberty-Enabled` and be defined as using Augmented BNF as
1042 specified in section 2 of [\[RFC 2616\]](#).

```
1043 Liberty-Enabled = "Liberty-Enabled" ":" LIB_Version [" ," 1#Extension]  
1044 LIB_Version = "LIBV" "=" 1*absoluteURI  
1045 ; any spaces or commas in the absoluteURI MUST be escaped as defined  
1046 in section 2.4 of \[RFC 2396\]  
1047 Extension = ExtName "=" ExtValue  
1048 ExtName = (["." host] | <any field-value but ".", ",", " or "=">) <any  
1049 field-value but "=" or ", ">  
1050 ExtValue = <any field-value but ", ">
```

1052 The comment, field-value, and product productions are defined in [\[RFC 2616\]](#). `LIB_Version`
1053 identifies the versions of the Liberty specifications that are supported by this LECP. Each version
1054 is identified by a URI. Service providers or identity providers receiving a Liberty-Enabled header
1055 MUST ignore any URIs listed in the `LIB_Version` production that they do not recognize. All
1056 LECPs compliant with this specification MUST send out, at minimum, the URI
1057 `http://projectliberty.org/specs/v1` as a value in the `LIB_Version` production. The
1058 ordering of the URIs in the `LIB_Version` header is meaningful; therefore, service providers and
1059 identity providers are encouraged to use the first version in the list that they support. Supported
1060 Liberty versions are not negotiated between the LECP and the service provider. The LECP simply
1061 advertises what version it does support, and the service provider MUST return the response for the
1062 corresponding version as defined in step 3 below.

1063 Optional extensions MAY be added to the Liberty-Enabled header to indicate new information.
1064 The value of the `ExtName` production MUST use the “host” “,” prefixed form if the new

1065 extension name has not been standardized and registered with Liberty or its designated registration
1066 authorities. The value of the `host` production MUST be an IP or DNS address that is owned by
1067 the issuer of the new name. By using the DNS/IP prefix, namespace collisions can be effectively
1068 prevented without having to introduce yet another centralized registration agency.

1069 LECPs MAY include the Liberty-Agent header in their requests. This header provides information
1070 about the software implementing the LECP functionality and is similar to the User-Agent and
1071 Server headers in HTTP.

```
1072 Liberty-Agent = "Liberty-Agent" ":" 1*( product | comment)  
1073
```

1074 Note: The reason for introducing the new header (that is, Liberty-Enabled) rather
1075 than just using User-Agent is that LECP may be a Liberty-enabled proxy. In that
1076 case the information about the Liberty-enabled proxy would not be in the User-
1077 Agent header. In theory the information could be in the VIA header. However, for
1078 security reasons, values in the VIA header can be collapsed, and comments (where
1079 software information would be recorded) can always be removed. As such, the
1080 VIA header is not suitable. Using the User-Agent header for a Liberty-enabled
1081 client and the Liberty-Agent header for a Liberty-enabled proxy was also
1082 discussed. However, this approach seemed too complex.

1083 Originally the Liberty-Agent header was going to be part of the Liberty-Enabled
1084 header. However, header lengths in HTTP implementations are limited; therefore,
1085 putting this information in its own header was considered the preferred approach.

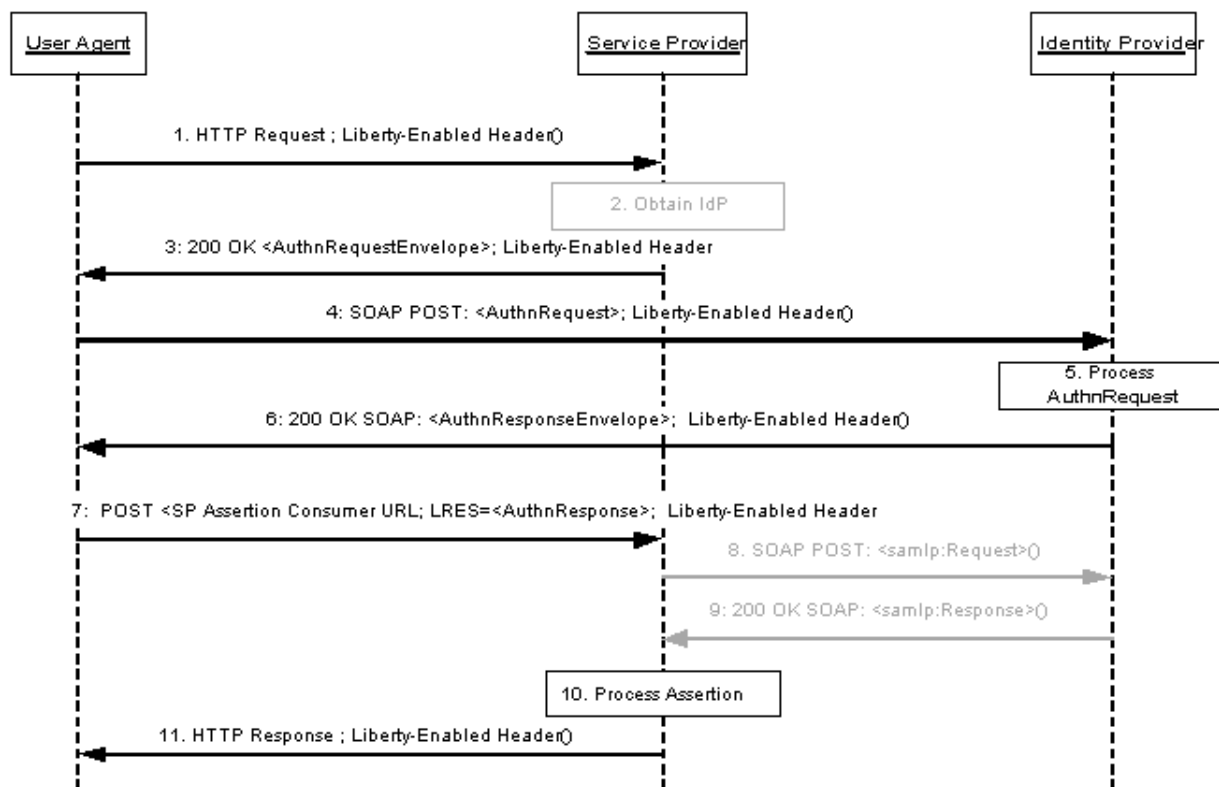
1086 A LECP MAY add a Liberty-Enabled entry in the HTTP User-Agent request header. The HTTP
1087 User-Agent header is specified in [\[RFC2616\]](#). A LECP MAY include in the value of this header
1088 the `Liberty-Enabled` string as defined above for the Liberty-Enabled header.

1089 Note: The reason for adding information to the User-Agent header is to allow for
1090 Liberty-enabled client products that must rely on a platform that cannot be
1091 instructed to insert new headers in each HTTP request.

1092 The User-Agent header is often overloaded; therefore, the Liberty-Enabled header
1093 should be the first choice for any implementation of a LECP. The entry in the
1094 User-Agent header then remains as a last resort.

1095 **3.2.5.2 Interactions**

1096 Figure 5 illustrates the Liberty-enabled client and proxy profile for single sign-on.



1097

1098

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

1099 This profile description assumes that the user agent has already authenticated at the identity
1100 provider prior to step 1. Thus, a valid session exists for the user agent at the identity provider.

1101 The LECP receives authentication requests from the service provider in the body of the HTTP
1102 response. The LECP submits this authentication request as a SOAP request to the identity
1103 provider. Because this SOAP request is between the LECP and the identity provider, TLS
1104 authentication cannot be performed between service provider and identity provider; therefore,
1105 service providers and identity providers MUST rely on the signature of the
1106 <lib:AuthnRequest> and the returned <saml:Assertion>, respectively, for mutual
1107 authentication.

1108 When implementing this profile, processing rules for steps 5, 10, and 11 defined in 3.2.1 for single
1109 sign-on profiles MUST be followed, while steps 2, 8, and 9 MUST be omitted. Additionally, the
1110 following rules MUST be observed as they relate to steps 1, 3, 4, 6, and 7:

1111 **Step 1: Accessing the Service Provider**

1112 In step 1, the user agent accesses the service provider with the Liberty-Enabled header (or with the
1113 Liberty-Enabled entry in the User-Agent header) included in the HTTP request.

1114 The HTTP request MUST contain only one Liberty-Enabled header. Hence if a proxy receives a
1115 HTTP request that contains a Liberty-Enabled header, it MUST NOT add another Liberty-Enabled
1116 header. However, a proxy MAY replace the Liberty-Enabled header. A proxy that replaces or adds

1117 a Liberty-Enabled header MUST process `<lib:AuthnRequest>` messages as defined in steps 3
1118 and 4 as well as `<lib:AuthnResponse>` messages as specified in steps 6 and 7.

1119 It is RECOMMENDED that a LECP add “`application/vnd.liberty-request+xml`” as one
1120 of its supported content types to the Accept header.

1121 **Step 3: HTTP Response with `<AuthnRequest>`**

1122 In step 3, the service provider’s intersite transfer service issues an HTTP 200 OK response to the
1123 user agent. The response MUST contain a single `<lib:AuthnRequestEnvelope>` with content
1124 as defined in [[LibertyProtSchema](#)]. If a service provider receives a Liberty-Enabled header, or a
1125 User-Agent header with the Liberty-Enabled entry, the service provider MUST respond according
1126 to the Liberty-enabled client and proxy profile and include a `Liberty_Enabled` header in its
1127 response. Hence service providers MUST support the Liberty-enabled client and proxy profile.

1128 The processing rules and default values for the Liberty-Enabled indications are as defined in
1129 3.2.5.1. The service provider MAY advertise any Liberty version supported in this header, not only
1130 the version used for the specific response.

1131 The HTTP response MUST contain a Content-Type header with the value
1132 `application/vnd.liberty-request+xml` unless the LECP and service provider have
1133 negotiated a different format.

1134 A service provider MAY provide a list of identity providers it recognizes by including the
1135 `<lib:IDPList>` element in the `<lib:AuthnRequestEnvelope>`. The format and processing
1136 rules for the identity provider list MUST be as defined in [[LibertyProtSchema](#)].

1137 Note: In cases where a value for the `<lib:GetComplete>` element is provided
1138 within `<lib:IDPList>`, the URI value for this element MUST specify `https` as
1139 the URL `<scheme>`.

1140 The service provider MUST specify a URL for receiving `<AuthnResponse>` elements, locally
1141 generated by the intermediary, by including the `<lib:AssertionConsumerServiceURL>`
1142 element in the `<lib:AuthnRequestEnvelope>`.

1143 The following example demonstrates the usage of the `<lib:AuthnRequestEnvelope>`:

```
1144 <?xml version="1.0" ?>  
1145 <lib:AuthnRequestEnvelope xmlns:lib="http://projectliberty.org/schemas/core/2002/12/">  
1146 <lib:AuthnRequest >  
1147 . . . AuthnRequest goes here . . .  
1148 </lib:AuthnRequest>  
1149 <lib:AssertionConsumerServiceURL>  
1150 https://service-provider.com/LibertyLogin  
1151 </lib:AssertionConsumerServiceURL>  
1152 <lib:IDPList >  
1153 . . . IdP list goes here . . .  
1154 </lib:IDPList>  
1155 </lib:AuthnRequestEnvelope>
```

1156

1157 If the service provider does not support the LECP-advertised Liberty version, the service provider
1158 MUST return to the LECP an HTTP 501 response with the reason phrase “Unsupported Liberty
1159 Version.”

1160 The responses in step 3 and step 6 SHOULD NOT be cached. To this end service providers and
1161 identity providers SHOULD place both “`Cache-Control: no-cache`” and “`Pragma: no-`

1162 cache” on their responses to ensure that the LECP and any intervening proxies will not cache the
1163 response.

1164 **Step 4: HTTP Request with <AuthnRequest>**

1165 In step 4, the LECP determines the appropriate identity provider to use and then issues an HTTP
1166 POST of the <lib:AuthnRequest> in the body of a SOAP message to the identity provider’s
1167 single sign-on service URL. The request **MUST** contain the same <lib:AuthnRequest> as was
1168 received in the <lib:AuthnRequestEnvelope> from the service provider in step 3.

1169 Note: The identity provider list can be used by the LECP to create a user identifier
1170 to be presented to the Principal. For example, the LECP could compare the list of
1171 the Principal’s known identities (and the identities of the identity provider that
1172 provides those identities) against the list provided by the service provider and then
1173 only display the intersection.

1174 If the LECP discovers a syntax error due to the service provider or cannot proceed any further for
1175 other reasons (for example, cannot resolve identity provider, cannot reach the identity provider,
1176 etc), the LECP **MUST** return to the service provider a <lib:AuthnResponse> with a
1177 <samlp:Status> indicating the desired error element as defined in [[LibertyProtSchema](#)]. The
1178 <lib:AuthnResponse> containing the error status **MUST** be sent using a POST to the service
1179 provider’s assertion consumer service URL obtained from the
1180 <lib:AssertionConsumerServiceURL> element of the <lib:AuthnRequestEnvelope>.
1181 The POST **MUST** be a form that contains the field LARES with the value being the
1182 <lib:AuthnResponse> protocol message as defined in [[LibertyProtSchema](#)], containing the
1183 <samlp:Status>. The <lib:AuthnResponse> **MUST** be encoded by applying a base64
1184 transformation (refer to
1185 [[RFC2045](#)]) to the <lib:AuthnResponse> and all its elements.

1186 **Step 6: HTTP Response with <AuthnResponse>**

1187 In step 6, the identity provider responds to the <lib:AuthnRequest> by issuing an HTTP 200
1188 OK response. The response **MUST** contain a single <lib:AuthnResponseEnvelope> in the
1189 body of a SOAP message with content as defined in [[LibertyProtSchema](#)].

1190 The identity provider **MUST** include the Liberty-Enabled HTTP header following the same
1191 processing rules as defined in 3.2.5.1.

1192 The Content-Type **MUST** be set to `application/vnd.liberty-response+xml`.

1193 If the identity provider discovers a syntax error due to the service provider or LECP or cannot
1194 proceed any further for other reasons (for example, unsupported Liberty version), the identity
1195 provider **MUST** return to the LECP a <lib:AuthnResponseEnvelope> containing a
1196 <lib:AuthnResponse> with a <samlp:Status> indicating the desired error element as
1197 defined in [[LibertyProtSchema](#)].

1198 **Step 7: Posting the Form Containing the <AuthnResponse>**

1199 In step 7, the LECP issues an HTTP POST of the <lib:AuthnResponse> that was received in
1200 the <lib:AuthnResonseEnvelope> SOAP response in step 6. The <lib:AuthnResponse>
1201 **MUST** be sent using a POST to the service provider’s assertion consumer service URL identified

1202 by the `<lib:AssertionConsumerServiceURL>` element within the
1203 `<lib:AuthnResponseEnvelope>` obtained from the identity provider in step 6. The POST
1204 MUST be a form that contains the field LARES with the value being the `<lib:AuthnResponse>`
1205 protocol message as defined in [[LibertyProtSchema](#)]. The `<lib:AuthnResponse>` MUST be
1206 encoded by applying a base64 transformation (refer to [[RFC2045](#)]) to the
1207 `<lib:AuthnResponse>` and all its elements. The service provider's assertion consumer service
1208 URL used as the target of the form POST MUST specify `https` as the URL scheme; if another
1209 scheme is specified, it MUST be treated as an error by the identity provider.

1210 If the LECP discovers an error (for example, syntax error in identity provider response), the LECP
1211 MUST return to the service provider a `<lib:AuthnResponse>` with a `<samlp:Status>`
1212 indicating the appropriate error element as defined in [[LibertyProtSchema](#)]. The
1213 `<lib:AuthnResponse>` containing the error status MUST be sent using a POST to the service
1214 provider's assertion consumer service URL. The POST MUST be a form that contains the field
1215 named LARES with its value being the `<lib:AuthnResponse>` protocol message as defined in
1216 [[LibertyProtSchema](#)] with formatting as specified 3.1.2. Any `<lib:AuthnResponse>` messages
1217 created by the identity provider MUST not be sent to the service provider.

1218 **3.3 Register Name Identifier Profiles**

1219 This section defines the profiles by which a provider may register or change a name identifier for a
1220 Principal. This message exchange is optional. During federation, the identity provider supplies an
1221 opaque handle identifying the Principle. This is the `<lib:IDPProvidedNameIdentifier>`. If
1222 neither provider involved in the federation opts to register any other name identifier, then this
1223 initial `<lib:IDPProvidedNameIdentifier>` is to be used by both providers.

1224 An identity provider may choose to register a new `<lib:IDPProvidedNameIdentifier>` at
1225 any time subsequent to federation, using this protocol. Additionally, a service provider may choose
1226 to register an `<lib:SPPProvidedNameIdentifier>`, which it expects the identity provider to
1227 use (instead of the `<lib:IDPProvidedNameIdentifier>`) when communicating with it about
1228 the Principal.

1229 Two profiles are specified: HTTP-Redirect-Based and SOAP/HTTP-based.

1230 Either the identity or service provider may initiate the register name identifier protocol. The
1231 available profiles are defined in 3.3.1 and 3.3.2, and vary slightly based on whether the protocol
1232 was initiated by the identity or service provider:

- 1233 • Register Name Identifier Initiated at Identity Provider
 - 1234 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between
 - 1235 the identity provider and the service provider.
 - 1236 • **SOAP/HTTP-Based:** Relies on a SOAP call from the identity provider to the
 - 1237 service provider.
- 1238 • Register Name Identifier Initiated at Service Provider
 - 1239 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between
 - 1240 the service provider and the identity provider.

- 1241 • **SOAP/HTTP-Based:** Relies on a SOAP call from the service provider to the
1242 identity provider.

1243 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles
1244 are essentially the same regardless of whether the profile was initiated at the service provider or at
1245 the identity provider, but the message flow directions are reversed.

1246 The register name identifier profiles make use of the following metadata elements, as defined in
1247 [[LibertyProtSchema](#)]:

- 1248 • `RegisterNameIdentifierProtocolProfile`: The service provider's preferred
1249 register name identifier profile, which should be used by the identity provider when
1250 registering a new identifier. This would specify the URI based identifier for one of
1251 the IDP Initiated register name identifier profiles.
- 1252 • `RegisterNameIdentifierServiceURL`: The URL used for user-agent-based
1253 Register Name Identifier Protocol profiles.
- 1254 • `RegisterNameIdentifierServiceReturnURL`: The provider's redirecting URL
1255 for use after HTTP name registration has taken place.
- 1256 • `SOAPEndpoint`: The SOAP endpoint location at the service provider or identity
1257 provider to which Liberty SOAP messages are sent.

1258 **3.3.1 Register Name Identifier Initiated at Identity Provider**

1259 An identity provider MAY change the `<lib:IDPProvidedNameIdentifier>` it has assigned a
1260 Principal and transmit that information to a service provider. The
1261 `<lib:IDPProvidedNameIdentifier>` MAY be changed without changing any federations.
1262 The reasons an identity provider may wish to change the name identifier for a Principal are
1263 implementation dependent, and thus outside the scope of this specification. Changing the
1264 `<lib:IDPProvidedNameIdentifier>` MAY be accomplished in either an HTTP-Redirect-
1265 Based or SOAP/HTTP mode.

1266 **3.3.1.1 HTTP-Redirect-Based Profile**

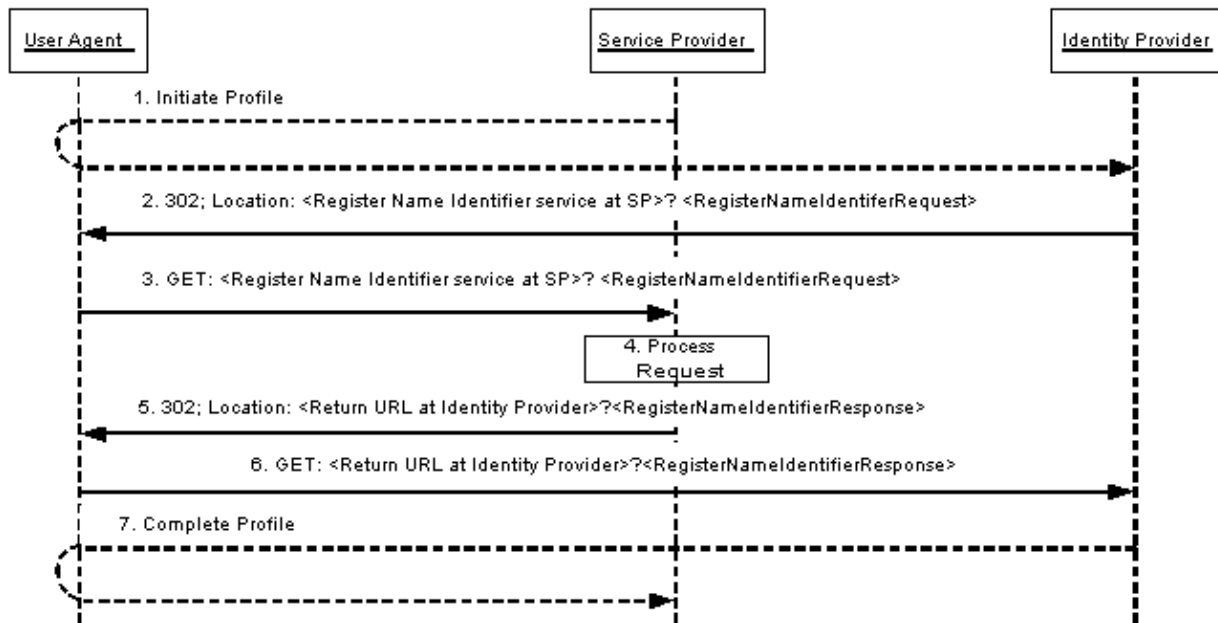
1267 A HTTP-redirect-based register name identifier profile cannot be self-initiated by an identity
1268 provider, but must be a triggered by a message, such as an `<lib:AuthnRequest>`. We note that
1269 we do not normatively specify when and how the identity provider can initiate this profile—that is
1270 left to the discretion of the identity provider. As an example, it may be triggered by a message,
1271 such as an `<lib:AuthnRequest>`. When the identity provider decides to initiate the profile in
1272 this case, it will insert this profile between the `AuthnRequest/AuthnResponse` transactions.

1273 The HTTP-redirect-based profile relies on using HTTP 302 redirects to communicate register
1274 name identifier messages from the identity provider to the service provider. The HTTP-Redirect
1275 Register Name Identifier Profile (Figure 6) illustrates this transaction.

1276 The following URI-based identifier MUST be used when referencing this specific profile:

1277 URI: <http://projectliberty.org/profiles/rni-idp-http>

1278 This URI identifier **MUST** be specified in the service provider metadata element
 1279 `RegisterNameIdentifierProtocolProfile` when the service provider intends to indicate to
 1280 the identity provider a preference for receiving register name identifier messages via a HTTP 302
 1281 redirect.



1282

1283

Figure 6. Register Name Identifier Profile.

1284 In an example scenario, the service provider makes an `<lib:AuthnRequest>` to the identity
 1285 provider for authentication of the Principal’s User Agent (step 1). The identity provider effects an
 1286 `<lib:IDPProvidedNameIdentifier>` change in the service provider via a URL redirection.
 1287 The profile is as follows:

1288 **Step 1: Initiate Profile**

1289 This interaction is not normatively specified as part of the profile, but shown for illustrative
 1290 purposes.

1291 **Step 2: Redirecting to the Service Provider Register Name Identifier Service**

1292 In step 2, the identity provider redirects the user agent to the register name identifier service at the
 1293 service provider.

1294 The redirection **MUST** adhere to the following rules:

- 1295 • The Location HTTP header **MUST** be set to the service provider’s register name
 1296 identifier service URL.

1297 • The service provider’s register name identifier service URL MUST specify https as the
1298 URL scheme; if another scheme is specified, the identity provider MUST NOT
1299 redirect to the service provider.

1300 • The Location HTTP header MUST include a <query> component containing the
1301 <lib:RegisterNameIdentifierRequest> protocol message as defined in
1302 [\[LibertyProtSchema\]](#) with formatting as specified in 3.1.2.

1303 The HTTP response MUST take the following form:

```
1304 <HTTP-Version> 302 <Reason Phrase>  
1305 <other headers>  
1306 Location : https://<Service Provider Register Name Identifier service URL>?<query>  
1307 <other HTTP 1.0 or 1.1 components>
```

1308 where

1309 <Service Provider Register Name Identifier service URL>

1310 This element provides the host name, port number, and path components of the register name
1311 identifier service URL at the service provider.

1312 <query>= ...<URL-encoded RegisterNameIdentifierRequest>...

1313 The <query> component MUST contain a single register name identifier request.

1314 **Step 3: Accessing the Service Provider Register Name Identifier Service**

1315 In step 3, the user agent accesses the service provider’s register name identifier service URL with
1316 the <lib:RegisterNameIdentifierRequest> information attached to the URL fulfilling the
1317 redirect request.

1318 **Step 4: Processing the Register Name Identifier Request**

1319 In step 4, the service provider MUST process the <lib:RegisterNameIdentifierRequest>
1320 according to the rules defined in [\[LibertyProtSchema\]](#).

1321 The service provider MAY remove the old name identifier after registering the new name
1322 identifier.

1323 **Step 5: Redirecting to the Identity Provider return URL with the Register Name 1324 Identifier Response**

1325 In step 5, the service provider’s register name identifier service responds and redirects the user
1326 agent back to identity provider using a return URL location specified in the
1327 RegisterNameIdentifierServiceReturnURL metadata element. If the URL-encoded
1328 <lib: RegisterNameIdentifierRequest> message received in step 3 contains a parameter
1329 named RelayState, then the service provider MUST include a <query> component containing
1330 the same RelayState parameter and its value in its response to the identity provider.

1331 The redirection MUST adhere to the following rules:

1332 • The Location HTTP header MUST be set to the identity providers return URL
1333 specified in the RegisterNameIdentifierServiceReturnURL metadata
1334 element.

- 1335 • The identity provider's return URL MUST specify https as the URL scheme; if another
1336 scheme is specified, the service provider MUST NOT redirect to the identity
1337 provider.
- 1338 • The Location HTTP header MUST include a <query> component containing the
1339 <lib:RegisterNameIdentifierResponse> protocol message as defined in
1340 [\[LibertyProtSchema\]](#) with formatting as specified in 3.1.2.

1341 The HTTP response MUST take the following form:

```
1342 <HTTP-Version> 302 <Reason Phrase>  
1343 <other headers>  
1344 Location : https://<Identity Provider Service Return URL >?<query>  
1345 <other HTTP 1.0 or 1.1 components>
```

1346 where:

1347 <Identity Provider Service Return URL>

1348 This element provides the host name, port number, and path components of the return URL at
1349 the identity provider.

1350 <query>= ...<URL-encoded RegisterNameIdentifierResponse>...

1351 The <query> component MUST contain a single register name identifier response. The
1352 <URL-encoded RegisterNameIdentifierResponse> component MUST contain the
1353 identical RelayState parameter and its value that was received in the URL-encoded register
1354 name identifier message obtained in step 3. If no RelayState parameter was provided in the
1355 step 3 message, then a RelayState parameter MUST NOT be specified in the <URL-
1356 encoded RegisterNameIdentifierResponse>.

1357 **Step 6: Accessing the Identity Provider return URL with the Register Name** 1358 **Identifier Response**

1359 In step 6, the user agent accesses the identity provider's return URL location fulfilling the
1360 redirect request.

1361 **Step 7: Complete profile**

1362 This concludes the initial sequence, which triggered the initiation of this profile.

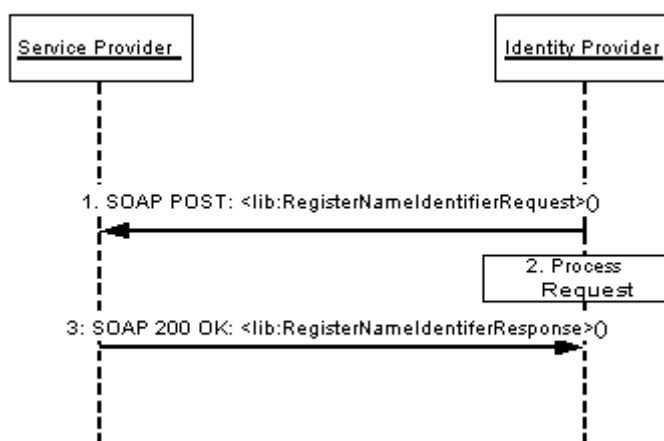
1363 **3.3.1.2 SOAP/HTTP-Based Profile**

1364 The following URI-based identifier MUST be used when referencing this specific profile:

1365 URI: `http://projectliberty.org/profiles/rni-idp-soap`

1366 This URI identifier MUST be specified in the service provider metadata element
1367 RegisterNameIdentifierProtocolProfile when the service provider intends to indicate to
1368 the identity provider a preference for receiving register name identifier messages via SOAP over
1369 HTTP.

1370 The steps involved in the SOAP/HTTP-based profile MUST utilize the SOAP binding for Liberty
1371 as defined in 2.1. See Figure 7.



1372

1373

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

1374 **Step 1: Request to Register Name Identifier**

1375 In step 1, the identity provider sends a `<lib:RegisterNameIdentifierRequest>` protocol
1376 message to the service provider's SOAP endpoint specifying
1377 `<lib:SPProvidedNameIdentifier>`, `<lib:IDPProvidedNameIdentifier>`, and
1378 `<lib:OldProvidedNameIdentifier>` as defined in [[LibertyProtSchema](#)]. The
1379 `<lib:SPProvidedNameIdentifier>` will only contain a value if the service provider has
1380 previously used the register name identifier profile.

1381 **Step 2: Process Request**

1382 Service provider records new `<lib:IDPProvidedNameIdentifier>`.

1383 **Step 3: Response to Register Name Identifier**

1384 The service provider, after successfully registering the new
1385 `<lib:IDPProvidedNameIdentifier>` provided by the identity provider, MUST respond with
1386 a `<lib:RegisterNameIdentifierResponse>` according to the processing rules defined in
1387 [[LibertyProtSchema](#)].

1388 **3.3.2 Register Name Identifier Initiated at Service Provider**

1389 A service provider may register, or change a `<lib:SPProvidedNameIdentifier>` which is a
1390 name identifier it expects the identity provider to use when communicating with it about the
1391 Principal. Until it registers a `<lib:SPProvidedNameIdentifier>`, an identity provider will
1392 continue to use the current `<lib:IDPProvidedNameIdentifier>` when referring to the
1393 Principal.

1394 **3.3.2.1 HTTP-Redirect-Based Profile**

1395 The HTTP-redirect-based profile relies on the use of a HTTP 302 redirect to communicate a
1396 register name identifier message from the service provider to the identity provider.

1397 The following URI-based identifier **MUST** be used when referencing this specific profile:

1398 URI: `http://projectliberty.org/profiles/rni-sp-http`

1399 A HTTP-redirect-based register name identifier profile can be self-initiated by a service provider
1400 to change the `<lib:SPProvidedNameIdentifier>`. We note that we do not normatively
1401 specify when and how the service provider can initiate this profile—that is left to the discretion of
1402 the service provider. The HTTP-redirect-based profile relies on using HTTP 302 redirects to
1403 communicate register name identifier messages from the service provider to the identity provider.
1404 The service provider effects a `<lib:SPProvidedNameIdentifier>` change in the identity
1405 provider via a URL redirection. For a discussion of the interactions and processing steps, refer to
1406 3.3.1.1. When reviewing that profile, interchange all references to service provider and identity
1407 provider in the interaction diagram and processing steps 3-7.

1408 **3.3.2.2 SOAP/HTTP-Based Profile**

1409 The SOAP/HTTP-based profile relies on using SOAP over HTTP to communicate register name
1410 identifier messages from the service provider to the identity provider. For a discussion of the
1411 interactions and processing steps, refer to 3.3.1.2. When reviewing that profile, interchange all
1412 references to service provider and identity provider in the interaction diagram and processing
1413 steps.

1414 The following URI-based identifier **MUST** be used when referencing this specific profile:

1415 URI: `http://projectliberty.org/profiles/rni-sp-soap`

1416 **3.4 Identity Federation Termination Notification Profiles**

1417 The Liberty identity federation termination notification profiles specify how service providers and
1418 identity providers are notified of federation termination (also known as defederation). Note: Other
1419 means of federation termination are possible, such as federation expiration and termination of
1420 business agreements between service providers and identity providers. These means of federation
1421 termination are outside the scope of this specification.

1422 Identity federation termination can be initiated at either the identity provider or the service
1423 provider. The Principal **SHOULD** have been authenticated by the provider at which identity
1424 federation termination is being initiated. The available profiles are defined in 3.4.1 and 3.4.2,
1425 depending on whether the identity federation termination notification process was initiated at the
1426 identity provider or service provider:

- 1427
- **Federation Termination Notification Initiated at Identity Provider**
 - **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between the
1428 identity provider and the service provider.
 - **SOAP/HTTP-Based:** Relies on a SOAP call from the identity provider to the service
1429 provider.
- 1430
- 1431

- 1432 • **Federation Termination Notification Initiated at Service Provider**
- 1433 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate between the
- 1434 service provider and the identity provider.
- 1435 • **SOAP/HTTP-Based:** Relies on a SOAP call from the service provider to the identity
- 1436 provider.

1437 The interactions and processing rules for the SOAP/HTTP-based and HTTP-redirect-based profiles

1438 are essentially the same regardless of whether federation termination notification was initiated at

1439 the service provider or at the identity provider.

1440 The identity federation termination notification profiles make use of the following metadata

1441 elements, as defined in [[LibertyProtSchema](#)]:

- 1442 • `FederationTerminationServiceURL` — The URL at the service provider or identity
- 1443 provider to which identity federation termination notifications are sent. It is documented in
- 1444 these profiles as “federation termination service URL.”
- 1445 • `FederationTerminationServiceReturnURL` — The URL used by the service
- 1446 provider or identity provider when redirecting the user agent at the end of the federation
- 1447 termination notification profile process.
- 1448 • `FederationTerminationNotificationProtocolProfile` — Used by the identity
- 1449 provider to determine which federation termination notification profile **MUST** be used
- 1450 when communicating with the service provider.
- 1451 • `SOAPEndpoint` — The SOAP endpoint location at the service provider or identity
- 1452 provider to which Liberty SOAP messages are sent.

1453 **3.4.1 Federation Termination Notification Initiated at Identity Provider**

1454 The profiles in 3.4.1.1 and 3.4.1.2 are specific to identity federation termination when initiated at

1455 the identity provider. Effectively, when using these profiles, the identity provider is stating to the

1456 service provider that it will no longer provide the Principal’s identity information to the service

1457 provider and that the identity provider will no longer respond to any requests by the service

1458 provider on behalf of the Principal.

1459 **3.4.1.1 HTTP-Redirect-Based Profile**

1460 The HTTP-redirect-based profile relies on using HTTP 302 redirect to communicate federation

1461 termination notification messages from the identity provider to the service provider. See Figure 8.

1462 The following URI-based identifier **MUST** be used when referencing this specific profile:

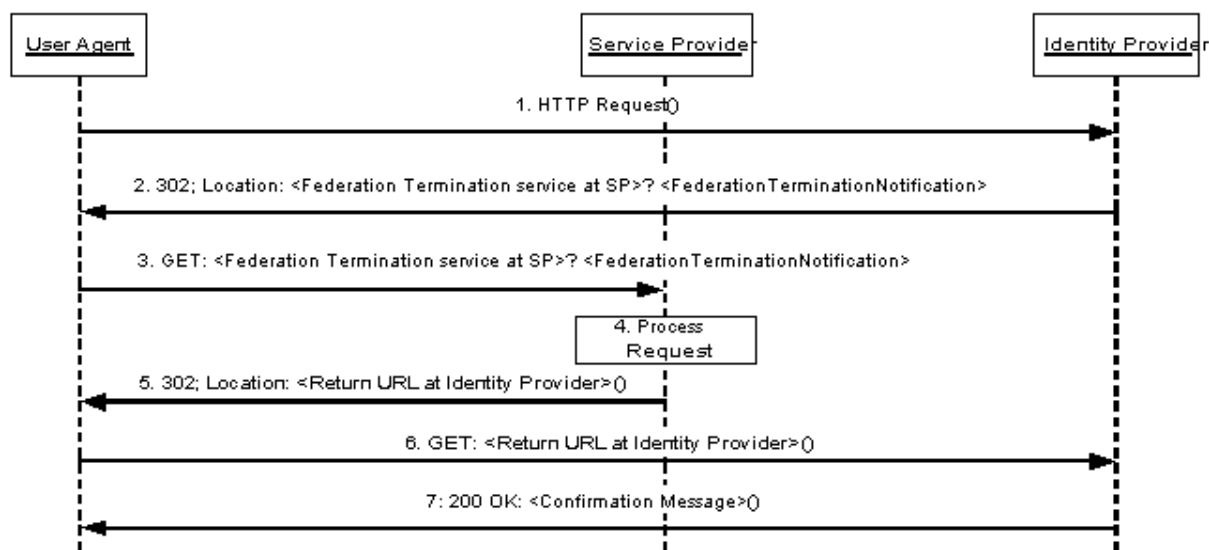
1463 URI: `http://projectliberty.org/profiles/fedterm-idp-http`

1464 This URI identifier **MUST** be specified in the service provider metadata element

1465 `FederationTerminationNotificationProtocolProfile` when the service provider

1466 intends to indicate to the identity provider a preference for receiving federation termination

1467 notifications via a HTTP 302 redirect.



1468

1469 **Figure 8: HTTP-redirect-based profile for federation termination**

1470 This profile description assumes the following preconditions:

- 1471 • The Principal’s identity at the service provider is federated with his/her identity at the
- 1472 identity provider.
- 1473 • The Principal has requested to the identity provider that the federation be terminated.
- 1474 • The Principal has authenticated with the identity provider.

1475 **Step 1: Accessing the Federation Termination Service**

1476 In step 1, the user agent accesses the identity federation termination service URL at the identity
 1477 provider specifying the service provider with which identity federation termination should occur.
 1478 How the service provider is specified is implementation-dependent and, as such, is out of the scope
 1479 of this specification.

1480 **Step 2: Redirecting to the Service Provider**

1481 In step 2, the identity provider’s federation termination service URL responds and redirects the
 1482 user agent to the federation termination service at the service provider.

1483 The redirection MUST adhere to the following rules:

- 1484 • The Location HTTP header MUST be set to the service provider’s federation termination
- 1485 service URL.
- 1486 • The service provider’s federation termination service URL MUST specify `https` as the
- 1487 URL scheme; if another scheme is specified, the identity provider MUST NOT redirect to
- 1488 the service provider.

- 1489 • The Location HTTP header MUST include a `<query>` component containing the
1490 `<lib:FederationTerminationNotification>` protocol message as defined in
1491 [\[LibertyProtSchema\]](#) with formatting as specified in 3.1.2.

1492 The HTTP response MUST take the following form:

```
1493 <HTTP-Version> 302 <Reason Phrase>  
1494 <other headers>  
1495 Location : https://<Service Provider Federation Termination service URL>?<query>  
1496 <other HTTP 1.0 or 1.1 components>
```

1497 where

1498 `<Service Provider Federation Termination service URL>`

1499 This element provides the host name, port number, and path components of the federation
1500 termination service URL at the service provider.

1501 `<query>= ...<URL-encoded FederationTerminationNotification>...`

1502 The `<query>` component MUST contain a single terminate federation request.

1503 **Step 3: Accessing the Service Provider Federation Termination Service**

1504 In step 3, the user agent accesses the service provider's federation termination service URL with
1505 the `<lib:FederationTerminationNotification>` information attached to the URL
1506 fulfilling the redirect request.

1507 **Step 4: Processing the Notification**

1508 In step 4, the service provider MUST process the

1509 `<lib:FederationTerminationNotification>` according to the rules defined in
1510 [\[LibertyProtSchema\]](#).

1511 The service provider MAY remove any locally stored references to the name identifier it received
1512 from the identity provider in the `<lib:FederationTerminationNotification>`.

1513 **Step 5: Redirecting to the Identity Provider Return URL**

1514 In step 5, the service provider's federation termination service responds and redirects the user
1515 agent back to identity provider using a return URL location specified in the
1516 `FederationTerminationServiceReturnURL` metadata element. If the URL-encoded
1517 `<lib:FederationTerminationNotification>` message received in step 3 contains a
1518 parameter named `RelayState`, then the service provider MUST include a `<query>` component
1519 containing the same `RelayState` parameter and its value in its response to the identity provider.

1520 No success or failure message should be conveyed in this HTTP redirect. The sole purpose of this
1521 redirect is to return the user agent to the identity provider where the federation termination process
1522 began.

1523 The HTTP response MUST take the following form:

```
1524 <HTTP-Version> 302 <Reason Phrase>  
1525 <other headers>  
1526 Location : https://<Identity Provider Service Return URL >?<query>  
1527 <other HTTP 1.0 or 1.1 components>
```

1528 where

1529 <Identity Provider Service Return URL>
1530 This element provides the host name, port number, and path components of the return URL at
1531 the identity provider.

1532 <query>= ...RelayState=<...>
1533 The <query> component MUST contain the identical RelayState parameter and its value
1534 that was received in the URL-encoded federation termination message obtained in step 3. If no
1535 RelayState parameter was provided in the step 3 message, then a RelayState parameter
1536 MUST NOT be specified in the <query> component.

1537 **Step 6: Accessing the Identity Provider Return URL**

1538 In step 6, the user agent accesses the identity provider’s return URL location fulfilling the redirect
1539 request.

1540 **Step 7: Confirmation**

1541 In step 7, the user agent is sent an HTTP response that confirms the requested action of identity
1542 federation termination with the specific service provider.

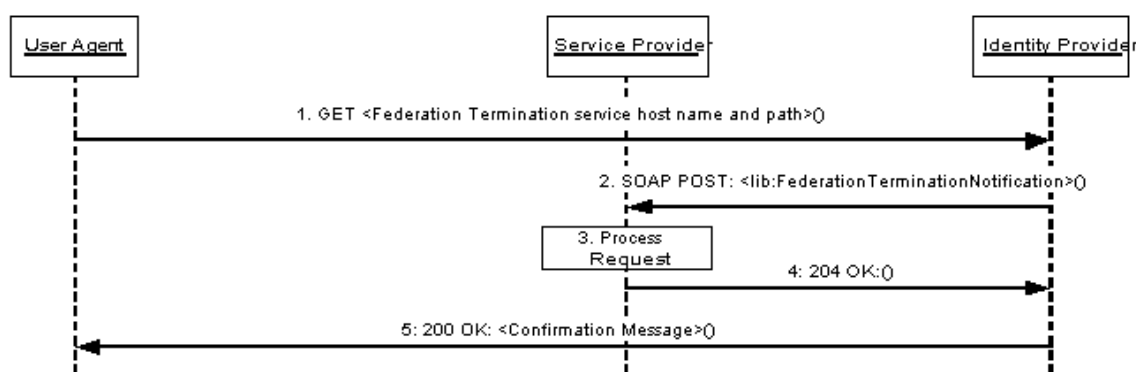
1543 **3.4.1.2 SOAP/HTTP-Based Profile**

1544 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate
1545 federation termination notification messages from the identity provider to the service provider.
1546 See Figure 9.

1547 The following URI-based identifier MUST be used when referencing this specific profile:

1548 URI: <http://projectliberty.org/profiles/fedterm-idp-soap>

1549 This URI identifier MUST be specified in the service provider metadata element
1550 FederationTerminationNotificationProtocolProfile when the service provider
1551 intends to indicate to the identity provider a preference for receiving federation termination
1552 notifications via SOAP over HTTP.



1553
1554 **Figure 9: SOAP/HTTP-based profile for federation termination**

1555 This profile description assumes the following preconditions:

- 1556 • The Principal's identity at the service provider is federated with his/her identity at the
1557 identity provider.
- 1558 • The Principal has requested to the identity provider that the federation be terminated.
- 1559 • The Principal has authenticated with the identity provider.

1560 **Step 1: Accessing the Federation Termination Service**

1561 In step 1, the user agent accesses the identity federation termination service URL at the identity
1562 provider specifying the service provider for with which identity federation termination should
1563 occur. How the service provider is specified is implementation-dependent and, as such, is out of
1564 the scope of this specification.

1565 **Step 2: Notification of Federation Termination**

1566 In step 2, the identity provider sends an asynchronous SOAP over HTTP notification message to
1567 the service provider's SOAP endpoint. The SOAP message **MUST** contain exactly one
1568 `<lib:FederationTerminationNotification>` element in the SOAP body and adhere to the
1569 construction rules defined in [[LibertyProtSchema](#)].

1570 If a SOAP fault occurs, the identity provider **SHOULD** employ best effort to resolve the fault
1571 condition and resend the federation termination notification message to the service provider.

1572 **Step 3: Processing the Notification**

1573 In step 3, the service provider **MUST** process the
1574 `<lib:FederationTerminationNotification>` according to the rules defined in
1575 [[LibertyProtSchema](#)].

1576 The service provider **MAY** remove any locally stored references to the name identifier it received
1577 from the identity provider in the `<lib:FederationTerminationNotification>`.

1578 **Step 4: Responding to the Notification**

1579 In step 4, the service provider **MUST** respond to the
1580 `<lib:FederationTerminationNotification>` with a HTTP 204 OK response.

1581 **Step 5: Confirmation**

1582 In step 5, the user agent is sent an HTTP response that confirms the requested action of identity
1583 federation termination with the specific service provider.

1584 **3.4.2 Federation Termination Notification Initiated at Service Provider**

1585 The profiles in 3.4.2.1 and 3.4.2.2 are specific to identity federation termination notification when
1586 initiated by a Principal at the service provider. Effectively, when using this profile, the service
1587 provider is stating to the identity provider that the Principal has requested that the identity provider
1588 no longer provide the Principal's identity information to the service provider and that service
1589 provider will no longer ask the identity provider to do anything on the behalf of the Principal.

1590 It is **RECOMMENDED** that the service provider, after initiating or receiving a federation
1591 termination notification, invalidate the local session for the Principal that was authenticated at the

1592 identity provider with which federation has been terminated. If the Principal was locally
1593 authenticated at the service provider, the service provider MAY continue to maintain a local
1594 session for the Principal. If the Principal wants to engage in a single sign-on session with identity
1595 provider again, the service provider MUST first federate with identity provider the given Principal.

1596 **3.4.2.1 HTTP-Redirect-Based Profile**

1597 The HTTP-redirect-based profile relies on the use of a HTTP 302 redirect to communicate a
1598 federation termination notification message from the service provider to the identity provider. For
1599 a discussion of the interactions and processing steps, refer to 3.4.1.1. When reviewing that profile,
1600 interchange all references to service provider and identity provider in the interaction diagram and
1601 processing steps.

1602 The following URI-based identifier MUST be used when referencing this specific profile:

1603 URI: `http://projectliberty.org/profiles/fedterm-sp-http`

1604 This URI identifier is really only meant for service provider consumption and as such is not
1605 needed in any provider metadata.

1606 **3.4.2.2 SOAP/HTTP-Based Profile**

1607 The SOAP/HTTP-based profile relies on using asynchronous SOAP over HTTP to communicate
1608 federation termination notification messages from the service provider to the identity provider. For
1609 a discussion of the interactions and processing steps, refer to 3.4.1.2. When reviewing that profile,
1610 interchange all references to service provider and identity provider in the interaction diagram and
1611 processing steps.

1612 The following URI-based identifier MUST be used when referencing this specific profile:

1613 URI: `http://projectliberty.org/profiles/fedterm-sp-soap`

1614 This URI identifier is really only meant for service provider consumption and as such is not
1615 needed in any provider metadata.

1616 **3.5 Single Logout Profiles**

1617 The single logout profiles synchronize session logout functionality across all sessions that were
1618 authenticated by a particular identity provider. The single logout can be initiated at either the
1619 identity provider or the service provider. In either case, the identity provider will then
1620 communicate a logout request to each service provider with which it has established a session for
1621 the Principal. The negotiation of which single logout profile the identity provider uses to
1622 communicate with each service provider is based upon the `SingleLogoutProtocolProfile`
1623 provider metadata element defined in [[LibertyProtSchema](#)].

1624 The available profiles are defined in 3.5.1 and 3.5.2, depending on whether the single logout is
1625 initiated at the identity provider or service provider:

- 1626 • **Single Logout Initiated at Identity Provider**

- 1627 • **HTTP-Based:** Relies on using either HTTP 302 redirects or HTTP GET requests to
1628 communicate logout requests from an identity provider to the service providers.

- 1629 • **SOAP/HTTP-Based:** Relies on SOAP over HTTP messaging to communicate logout
1630 requests from an identity provider to the service providers.

- 1631 • **Single Logout Initiated at Service Provider**
- 1632 • **HTTP-Redirect-Based:** Relies on a HTTP 302 redirect to communicate a logout
- 1633 request with the identity provider.
- 1634 • **SOAP/HTTP-Based:** Relies on SOAP over HTTP messaging to communicate a logout
- 1635 request from a service provider to an identity provider.

1636 The single logout profiles make use of the following metadata elements, as defined in

1637 [\[LibertyProtSchema\]](#).

- 1638 • `SingleLogoutServiceURL` — The URL at the service provider or identity provider to
- 1639 which single logout requests are sent. It is described in these profiles as “single logout
- 1640 service URL.”
- 1641 • `SingleLogoutServiceReturnURL` — The URL used by the service provider when
- 1642 redirecting the user agent to the identity provider at the end of the single logout profile
- 1643 process.
- 1644 • `SingleLogoutProtocolProfile` — Used by the identity provider to determine which
- 1645 single logout request profile **MUST** be used when communicating with the service
- 1646 provider.
- 1647 • `SOAPEndpoint` — The SOAP endpoint location at the service provider or identity
- 1648 provider to which Liberty SOAP messages are sent.

1649 **3.5.1 Single Logout Initiated at Identity Provider**

1650 The profiles in 3.5.1.1 through 3.5.1.2 are specific to a single logout when initiated by a user agent

1651 at the identity provider.

1652 **3.5.1.1 HTTP-Based Profile**

1653 The HTTP-based profile defines two possible implementations that an identity provider may use.

1654 The first implementation relies on using HTTP 302 redirects, while the second uses HTTP GET

1655 requests. The choice of implementation is entirely dependent upon the type of user experience the

1656 identity provider provides.

1657 The following URI-based identifier **MUST** be used when referencing either implementation for

1658 this specific profile:

1659 URI: `http://projectliberty.org/profiles/slo-idp-http`

1660 This URI identifier **MUST** be specified in the service provider metadata element

1661 `SingleLogoutProtocolProfile` when the service provider intends to indicate to the identity

1662 provider a preference for receiving logout requests via either a HTTP redirect or a HTTP GET.

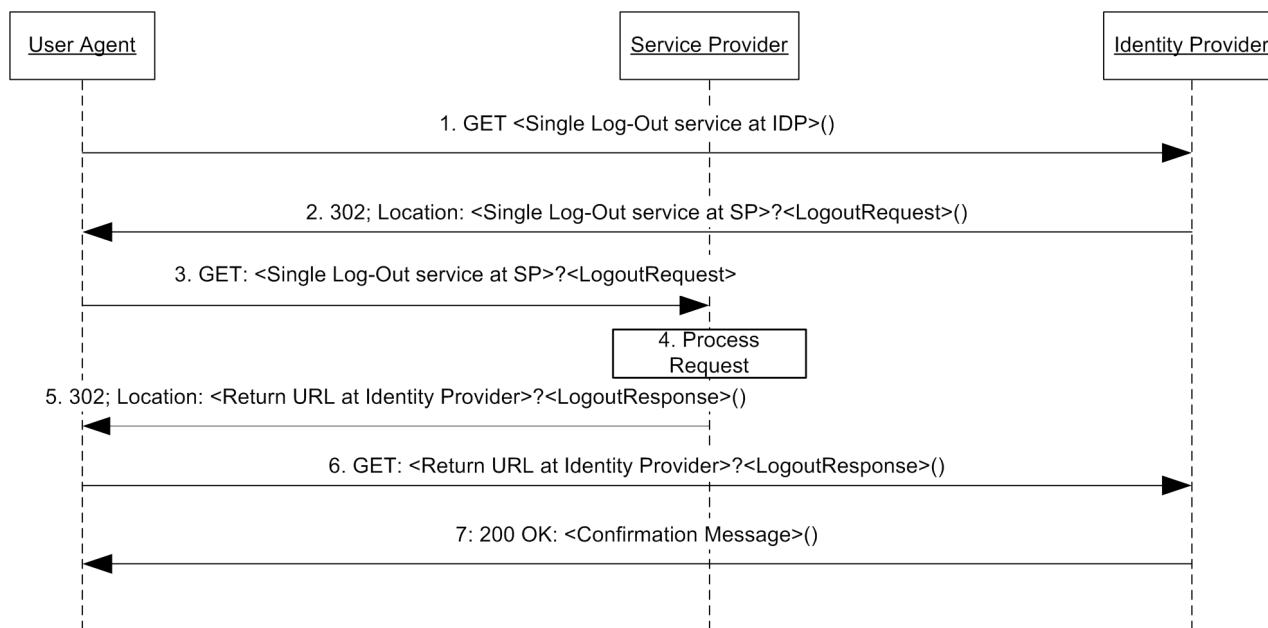
1663 **3.5.1.1.1 HTTP-Redirect Implementation**

1664 The HTTP-Redirect implementation uses HTTP 302 redirects to communicate a logout request to

1665 each service provider for which the identity provider has provided authentication assertions during

1666 the Principal’s current session if the service provider indicated a preference to receive logout

1667 requests via the HTTP based profile. See Figure 10.



1668

1669 **Figure 10: HTTP-Redirect implementation for single logout initiated at identity provider**

1670 Note: Steps 2 through 6 may be an iterative process for requesting logouts by each
 1671 service provider that has been issued authentication assertions during the
 1672 Principal’s current session and has indicated a preference to receive logout
 1673 requests via the HTTP based profile.

1674 Note: [RFC 2616](#) indicates a client should detect infinite redirection loops
 1675 because such loops generate network traffic for each redirection. This requirement
 1676 was introduced because previous versions of the specification recommended a
 1677 maximum of five redirections. Content developers should be aware that some
 1678 clients might implement such a fixed limitation.

1679 **Step 1: Accessing the Single Logout Service at the Identity Provider**

1680 In step 1, the user agent accesses the single logout service URL at the identity provider indicating
 1681 that all service providers for which this identity provider has provided authentication assertions
 1682 during the Principal’s current session must be notified of session termination.

1683 **Step 2: Redirecting to the Single Logout Service at the Service Provider**

1684 In step 2, the identity provider’s single logout service responds and redirects the user agent to the
 1685 single logout service URL at each service provider for which the identity provider has provided an
 1686 authentication assertion during the Principal’s current session with the identity provider.

1687 The redirections **MUST** adhere to the following rules:

- 1688 • The Location HTTP header **MUST** be set to the service provider’s single logout service
 1689 URL.

- 1690 • The service provider’s single logout service URL MUST specify `https` as the URL
1691 scheme; if another scheme is specified, the identity provider MUST NOT redirect to the
1692 service provider.
- 1693 • The Location HTTP header MUST include a `<query>` component containing the
1694 `<lib:LogoutRequest>` protocol message as defined in [[LibertyProtSchema](#)] with
1695 formatting as specified in 3.1.2.

1696 The HTTP response MUST take the following form:

```
1697 <HTTP-Version> 302 <Reason Phrase>  
1698 <other headers>  
1699 Location : https://<Service Provider Single Log-Out service URL>?<query>  
1700 <other HTTP 1.0 or 1.1 components>
```

1701 where

1702 `<Service Provider Single Log-Out service URL>`

1703 This element provides the host name, port number, and path components of the single logout
1704 service URL at the service provider.

1705 `<query>= ...<URL-encoded LogoutRequest>...`

1706 The `<query>` MUST contain a single logout request.

1707 **Step 3: Accessing the Service Provider Single Logout Service**

1708 In step 3, the user agent accesses the service provider’s single logout service URL with the
1709 `<lib:LogoutRequest>` information attached to the URL fulfilling the redirect request.

1710 **Step 4: Processing the Request**

1711 In step 4, the service provider MUST process the `<lib:LogoutRequest>` according to the rules
1712 defined in [[LibertyProtSchema](#)].

1713 The service provider MUST invalidate the session(s) of the Principal referred to in the name
1714 identifier it received from the identity provider in the `<lib:LogoutRequest>`.

1715 **Step 5: Redirecting to the Identity Provider Return URL**

1716 In step 5, the service provider’s single logout service responds and redirects the user agent back to
1717 the identity provider using the return URL location obtained from the
1718 `SingleLogoutServiceReturnURL` metadata element. If the URL-encoded
1719 `<lib:LogoutRequest>` message received in step 3 contains a parameter named `RelayState`,
1720 then the service provider MUST include a `<query>` component containing the same `RelayState`
1721 parameter and its value in its response to the identity provider.

1722 The purpose of this redirect is to return the user agent to the identity provider so that the single
1723 logout process may continue in the same fashion with other service providers.

1724 The HTTP response MUST take the following form:

```
1725 <HTTP-Version> 302 <Reason Phrase>  
1726 <other headers>  
1727 Location : https://<Identity Provider Service Return URL>?<query>  
1728 <other HTTP 1.0 or 1.1 components>
```

1729 where

1730 <Identity Provider Service Return URL>
1731 This element provides the host name, port number, and path components of the return URL at
1732 the identity provider.

1733 <query>= ...<URL-encoded LogoutResponse>
1734 The <query> component MUST contain a single logout response. The <URL-encoded
1735 LogoutResponse> MUST contain the identical RelayState parameter and its value that
1736 was received in the URL-encoded logout request message obtained in step 3. If no
1737 RelayState parameter was provided in the step 3 message, then a RelayState parameter
1738 MUST NOT be specified in the <URL-encoded LogoutResponse>.

1739 **Step 6: Accessing the Identity Provider Return URL**

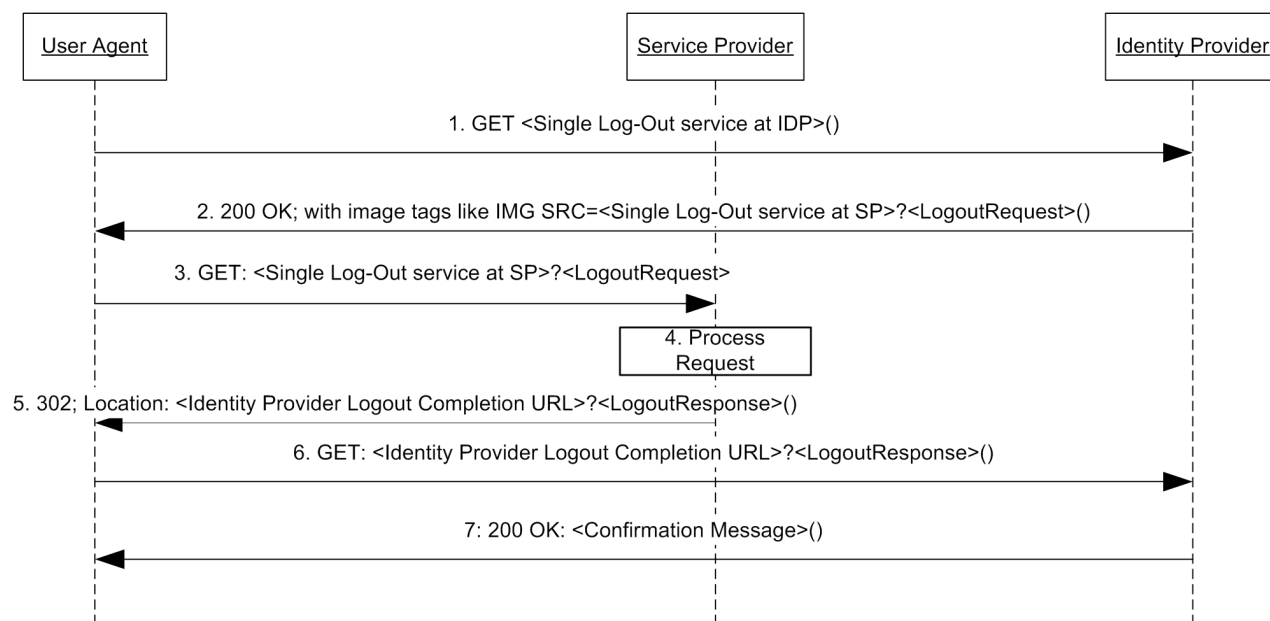
1740 In step 6, the user agent accesses the identity provider’s return URL location fulfilling the redirect
1741 request.

1742 **Step 7: Confirmation**

1743 In step 7, the user agent is sent an HTTP response that confirms the requested action of a single
1744 logout has been completed.

1745 **3.5.1.1.2 HTTP-GET Implementation**

1746 The HTTP-GET implementation uses HTTP GET requests to communicate logout requests to each
1747 service provider for which the identity provider has provided authentication during the Principal’s
1748 current session if the service provider indicated a preference to receive logout requests via the
1749 HTTP based profile. See Figure 11.



1750
1751 **Figure 11: HTTP-GET implementation for single logout initiated at identity provider**

1752 Note: Steps 3 through 7 may be an iterative process for requesting logout of each
1753 service provider that has been issued authentication assertions during the

1754 Principal's current session and has indicated a preference to receive logout
1755 requests via the HTTP based profile.

1756 **Step 1: Accessing the Single Logout Service at the Identity Provider**

1757 In step 1, the user agent accesses the single logout service URL at the identity provider indicating
1758 that all service providers for which this identity provider has provided authentication assertions
1759 during the Principal's current session must be notified of session termination and requested to
1760 logout the Principal.

1761 **Step 2: HTML Page Returned to User Agent with Image Tags**

1762 In step 2, the identity provider's single logout service responds with an HTML page that includes
1763 image tags referencing the logout service URL for each of the service providers for which the
1764 identity provider has provided an authentication assertion during the Principal's current session.
1765 The list of image tags MUST be sent in a standard HTTP 200 response to the user agent.

1766 The image tag loads on the HTML page MUST adhere to the following rules:

- 1767 • The SRC attribute MUST be set to the specific service provider's single logout service
1768 URL.
- 1769 • The service provider's single logout service URL MUST specify `https` as the URL
1770 scheme.
- 1771 • The service provider's single logout service URL MUST include a `<query>` component
1772 containing the `<lib:LogoutRequest>` protocol message as defined in
1773 [[LibertyProtSchema](#)] with formatting as specified in 3.1.2.

1774 **Step 3: Accessing the Service Provider Single Logout Service**

1775 In step 3, the user agent, as a result of each image load, accesses the service provider's single
1776 logout service URL with `<lib:LogoutRequest>` information attached to the URL. This step
1777 may occur multiple times if the HTTP response includes multiple image tag statements (one for
1778 each service provider that has been issued authentication assertions during the Principal's current
1779 session).

1780 **Step 4: Processing the Request**

1781 In step 4, the service provider MUST process the `<lib:LogoutRequest>` according to the rules
1782 defined in [[LibertyProtSchema](#)].

1783 The service provider MUST invalidate the session of the Principal referred to in the name
1784 identifier it received from the identity provider in the `<lib:LogoutRequest>`.

1785 **Step 5: Redirecting to the Identity Provider Logout Completion URL**

1786 In step 5, the service provider's single logout service responds and redirects the image load back to
1787 the identity provider's logout completion URL. This location will typically point to an image that
1788 will be loaded by the user agent to indicate that the logout is complete (for example, a checkmark).

1789 The logout completion URL is obtained from the `SingleLogoutServiceReturnURL` metadata
1790 element.

1791 The HTTP response MUST take the following form:

```
1792 <HTTP-Version> 302 <Reason Phrase>  
1793 <other headers>  
1794 Location : https://<Identity Provider Logout Completion URL>?<query>  
1795 <other HTTP 1.0 or 1.1 components>
```

1796 where

1797 <Identity Provider Logout Completion URL>

1798 This element provides the host name, port number, and path components of the identity
1799 provider logout completion URL at the identity provider.

1800 <query>=...<URL-encoded LogoutResponse>

1801 The <query> component MUST contain a single logout response. The <URL-encoded
1802 LogoutResponse> component MUST contain the identical `RelayState` parameter and its
1803 value that was received in the URL-encoded logout request message obtained in step 3. If no
1804 `RelayState` parameter was provided in step 3 then a `RelayState` message MUST NOT be
1805 specified in the <URL-encoded LogoutResponse>.

1806 **Step 6: Accessing the Identity Provider Logout Completion URL**

1807 In step 6, the user agent accesses the identity provider's logout completion URL fulfilling the
1808 redirect request.

1809 **Step 7: Confirmation**

1810 In step 7, the user agent is sent an HTTP response that confirms the requested action of a single
1811 logout has been completed.

1812 Note: One method for seamlessly returning the user agent back to the identity
1813 provider is for the HTML page generated in step 2 to include a script that runs
1814 when the page is completely loaded (all logouts completed) that will initiate the
1815 redirect to the identity provider.

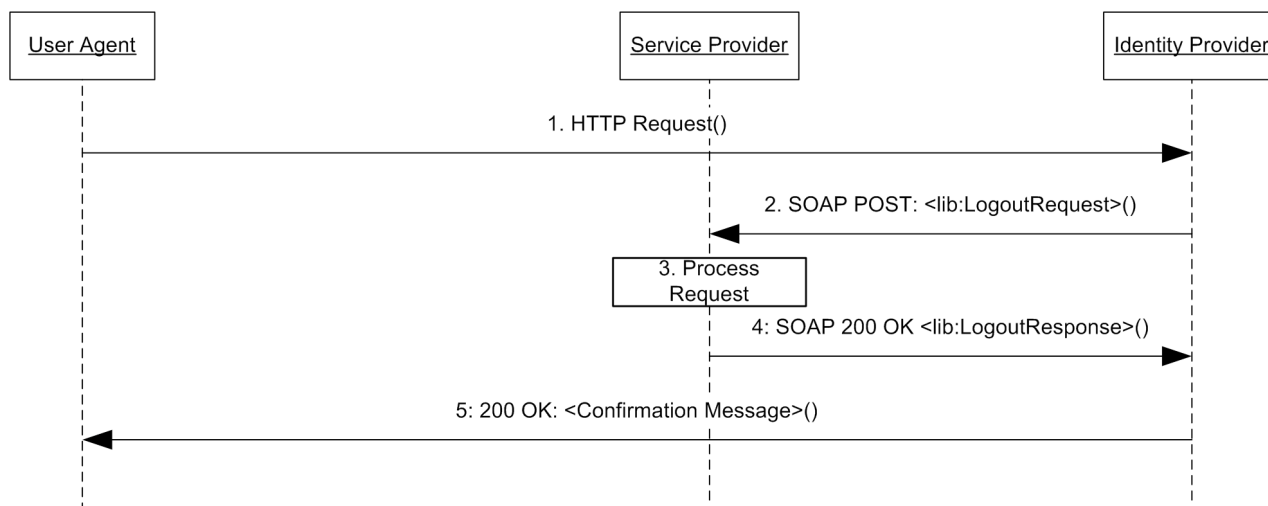
1816 **3.5.1.2 SOAP/HTTP-Based Profile**

1817 The SOAP/HTTP-based profile uses SOAP over HTTP messaging to communicate a logout
1818 request to each service provider for which the identity provider has provided authentication
1819 assertions during the Principal's current session if the service provider indicated a preference to
1820 receive logout request via the SOAP/HTTP-based profile. See Figure 12.

1821 The following URI-based identifier MUST be used when referencing this specific profile:

1822 URI: `http://projectliberty.org/profiles/slo-idp-soap`

1823 This URI identifier MUST be specified in the service provider metadata element
1824 `SingleLogoutProtocolProfile` when the service provider intends to indicate to the identity
1825 provider a preference for receiving logout requests via SOAP over HTTP.



1826

1827 **Figure 12: SOAP/HTTP-based profile for single logout initiated at identity provider**

1828 Note: Steps 2 through 4 may be an iterative process for each service provider that
 1829 has been issued authentication assertions during the Principal’s current session
 1830 and has indicated a preference to receive logout requests via the SOAP/HTTP
 1831 message profile.

1832 **Step 1: Accessing the Single Logout Service**

1833 In step 1, the user agent accesses the single logout service URL at the identity provider via an
 1834 HTTP request.

1835 **Step 2: Logout Request**

1836 In step 2, the identity provider sends a SOAP over HTTP request to the SOAP endpoint of each
 1837 service provider for which it provided authentication assertions during the Principal’s current
 1838 session. The SOAP message MUST contain exactly one <lib:LogoutRequest> element in the
 1839 SOAP body and adhere to the construction rules defined in [[LibertyProtSchema](#)].

1840 If a SOAP fault occurs, the identity provider SHOULD employ best efforts to resolve the fault
 1841 condition and resend the single logout request to the service provider.

1842 **Step 3: Processing the Logout Request**

1843 In step 3, the service provider MUST process the <lib:LogoutRequest> according to the rules
 1844 defined in [[LibertyProtSchema](#)].

1845 The service provider MUST invalidate the session for the Principal specified by the name
 1846 identifier provided by the identity provider in the <lib:LogoutRequest>.

1847 **Step 4: Responding to the Request**

1848 In step 4, the service provider MUST respond to the <lib:LogoutRequest> with a SOAP 200
 1849 OK <lib:LogoutResponse> message.

1850 **Step 5: Confirmation**

1851 In step 5, the user agent is sent an HTTP response that confirms the requested action of single
1852 logout has completed.

1853 **3.5.2 Single Logout Initiated at Service Provider**

1854 The profiles in 3.5.2.1 and 3.5.2.2 are specific to the Principal's initiation of the single logout
1855 request process at the service provider.

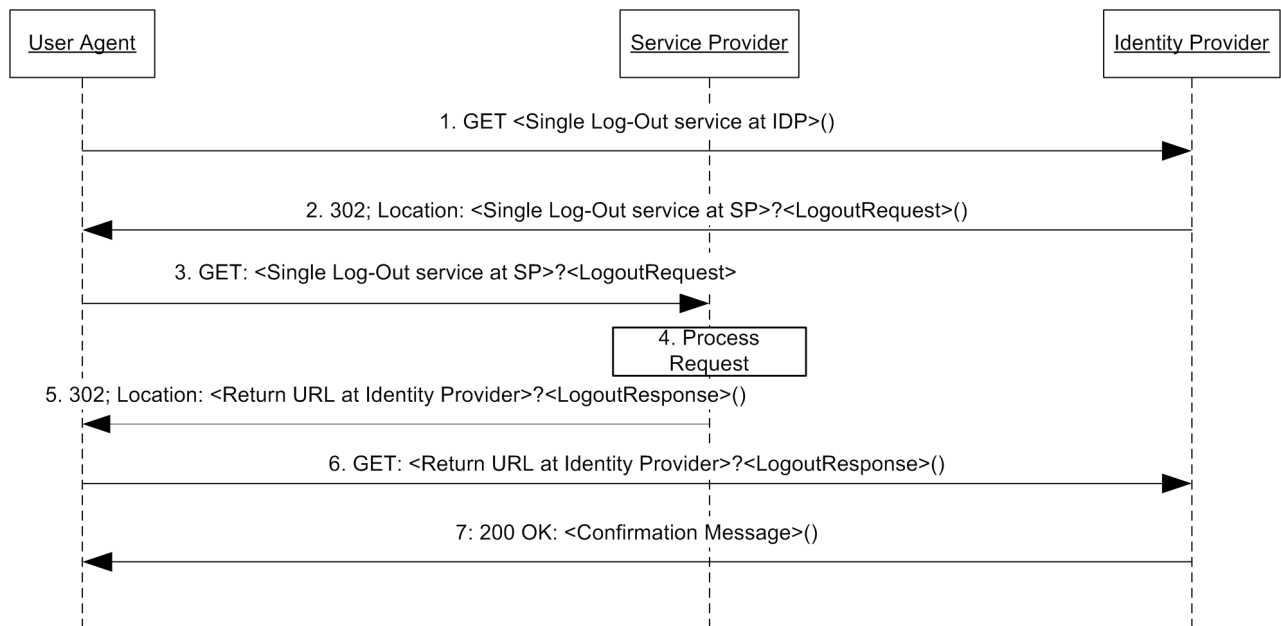
1856 **3.5.2.1 HTTP-Redirect-Based Profile**

1857 The HTTP-redirect-based profile relies on using a HTTP 302 redirect to communicate a logout
1858 request with the identity provider. The identity provider will then communicate a logout request to
1859 each service provider with which it has established a session for the Principal using the service
1860 provider's preferred profile for logout request from the identity provider (see 3.5.1). See Figure 13.

1861 The following URI-based identifier **MUST** be used when referencing this specific profile:

1862 URI: `http://projectliberty.org/profiles/slo-sp-http`

1863 This URI identifier is intended for service provider consumption and is not needed in provider
1864 metadata.



1865
1866 **Figure 13: HTTP-redirect-based profile for single logout initiated at service provider**

1867 Note: Step 4 may involve an iterative process by the identity provider to
1868 implement the preferred profile for logout requests for each service provider that
1869 has been issued authentication assertions during the Principal's current session.

1870 **Step 1: Accessing the Single Logout Service at the Service Provider**

1871 In step 1, the user agent accesses the single logout service URL at the service provider indicating
1872 that session logout is desired at the associated identity provider and all service providers for which
1873 this identity provider has provided authentication assertions during the Principal's current session.
1874 If a current session exists for the Principal at the service provider, it is RECOMMENDED that the
1875 service provider terminate that session prior to step 2.

1876 **Step 2: Redirecting to the Single Logout Service at the Identity Provider**

1877 In step 2, the service provider's single logout service responds and redirects the user agent to the
1878 single logout service URL at the identity provider.

1879 The redirection MUST adhere to the following rules:

- 1880 • The Location HTTP header MUST be set to the identity provider's single logout service
1881 URL.
- 1882 • The identity provider's single logout service URL MUST specify `https` as the URL
1883 scheme; if another scheme is specified, the service provider MUST NOT redirect to the
1884 identity provider.
- 1885 • The Location HTTP header MUST include a `<query>` component containing the
1886 `<lib:LogoutRequest>` protocol message as defined in [[LibertyProtSchema](#)] with
1887 formatting as specified in 3.1.2.

1888 The HTTP response MUST take the following form:

```
1889 <HTTP-Version> 302 <Reason Phrase>  
1890 <other headers>  
1891 Location : https://<Identity Provider single log-out service URL>?<query>  
1892 <other HTTP 1.0 or 1.1 components>
```

1893 where

1894 `<Identity Provider single log-out service URL>`

1895 This element provides the host name, port number, and path components of the single logout
1896 service URL at the identity provider.

1897 `<query>= ...<URL-encoded LogoutRequest>...`

1898 The `<query>` MUST contain a single logout request.

1899 **Step 3: Accessing the Identity Provider Single Logout Service**

1900 In step 3, the user agent accesses the identity provider's single logout service URL with the
1901 `<lib:LogoutRequest>` information attached to the URL fulfilling the redirect request.

1902 **Step 4: Processing the Request**

1903 In step 4, the identity provider MUST process the `<lib:LogoutRequest>` according to the rules
1904 defined in [[LibertyProtSchema](#)].

1905 Each service provider for which the identity provider has provided authentication assertions during
1906 the Principal's current session MUST be notified via the service provider's preferred profile for
1907 logout request from the identity provider (see 3.5.1).

1908 The identity provider's current session with the Principal MUST be terminated, and no more
1909 authentication assertions for the Principal are to be given to service providers.

1910 **Step 5: Redirecting to the Service Provider Return URL**

1911 In step 5, the identity provider's single logout service responds and redirects the user agent back to
1912 service provider using the return URL location obtained from the
1913 SingleLogoutServiceReturnURL metadata element. If the URL-encoded
1914 <lib:LogoutRequest> message received in step 3 contains a parameter named RelayState,
1915 then the identity provider MUST include a <query> component containing the same
1916 RelayState parameter and its value in its response to the service provider.

1917 The purpose of this redirect is to return the user agent to the service provider.

1918 The HTTP response MUST take the following form:

```
1919 <HTTP-Version> 302 <Reason Phrase>  
1920 <other headers>  
1921 Location : https://<Service Provider Return Service URL>?<query>  
1922 <other HTTP 1.0 or 1.1 components>
```

1923 where

1924 <Service Provider Service Return URL>

1925 This element provides the host name, port number, and path components of the return URL
1926 location at the service provider.

1927 <query>= ...<URL-encoded LogoutResponse>

1928 The <query> component MUST contain a single logout response. The <URL-encoded
1929 LogoutResponse> component MUST contain the identical RelayState parameter and its
1930 value that was received in the URL-encoded logout request message obtained in step 3. If no
1931 RelayState parameter was provided in the step 3 message, then a RelayState parameter
1932 MUST NOT be specified in the <URL-encoded LogoutResponse>.

1933 **Step 6: Accessing the Service Provider Return URL**

1934 In step 6, the user agent accesses the service provider's return URL location fulfilling the redirect
1935 request.

1936 **Step 7: Confirmation**

1937 In step 7, the user agent is sent an HTTP response that confirms the requested action of a single
1938 logout has been completed.

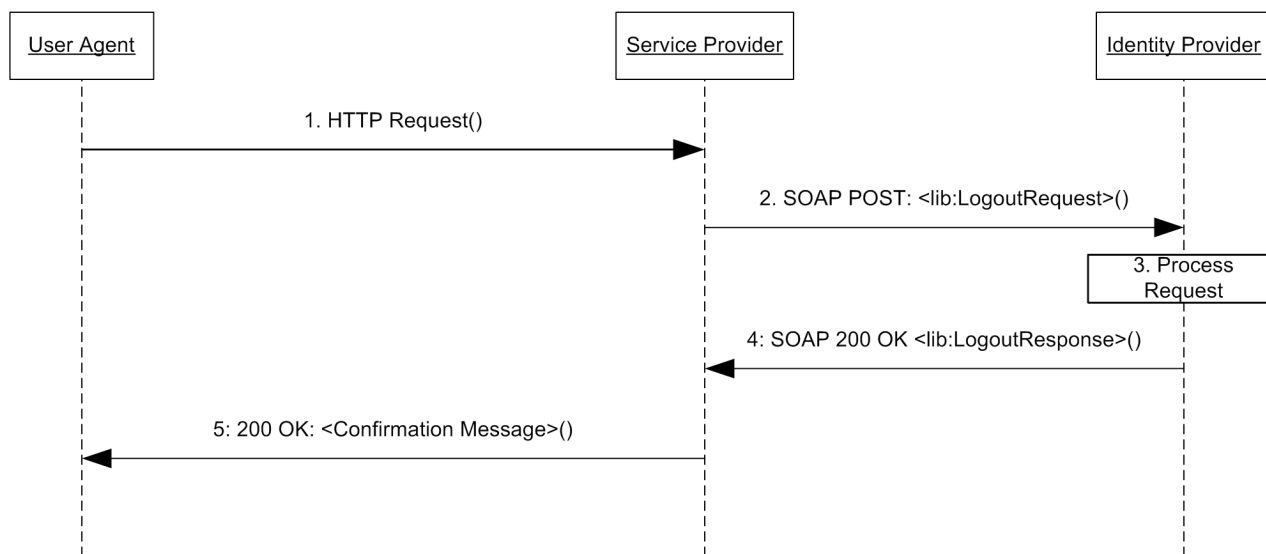
1939 **3.5.2.2 SOAP/HTTP-Based Profile**

1940 The SOAP/HTTP-based profile relies on using SOAP over HTTP messages to communicate a
1941 logout request to the identity provider. The identity provider will then communicate a logout
1942 request to each service provider it has established a session with for the Principal via the service
1943 provider's preferred profile for logout requests from the identity provider (see 3.5.1). See Figure
1944 14.

1945 The following URI-based identifier MUST be used when referencing this specific profile:

1946 URI: <http://projectliberty.org/profiles/slo-sp-soap>

1947 This URI identifier is intended for service provider consumption and is not needed in provider
1948 metadata.



1949
1950 **Figure 14: SOAP/HTTP-based profile for single logout initiated at service provider**

1951 Note: Step 3 may involve an iterative process by the identity provider to
1952 implement the preferred profile for logout requests for each service provider that
1953 has been issued authentication assertions during the Principal’s current session.

1954 **Step 1: Accessing Single Logout Service**

1955 In step 1, the user agent accesses the single logout service URL at the service provider via an
1956 HTTP request.

1957 **Step 2: Logout Request**

1958 In step 2, the service provider sends a SOAP over HTTP request to the identity provider’s SOAP
1959 endpoint. The SOAP message MUST contain exactly one <lib:LogoutRequest> element in the
1960 SOAP body and adhere to the construction rules as defined in [[LibertyProtSchema](#)].

1961 If a SOAP fault occurs, the service provider SHOULD employ best efforts to resolve the fault
1962 condition and resend the single logout request to the identity provider.

1963 **Step 3: Processing the Logout Request**

1964 In step 3, the identity provider MUST process the <lib:LogoutRequest> according to the rules
1965 defined in [[LibertyProtSchema](#)].

1966 Each service provider for which the identity provider has provided authentication assertions during
1967 the Principal’s current session MUST be requested to logout the Principal via the service
1968 provider’s preferred profile for logout requests from the identity provider. If the identity provider
1969 determines that one or more of service providers to which it has provided assertions regarding this
1970 Principal do not support the SOAP profiles for the single logout, the identity provider MUST
1971 return a <lib:LogoutResponse> containing a status code of <lib:UnsupportedProfile>.

1972 The service provider **MUST** then re-submit its LogoutRequest via the HTTP profile described
1973 above.

1974 The identity provider's current session with the Principal **MUST** be terminated, and no more
1975 authentication assertions for the Principal are to be given to service providers.

1976 **Step 4: Responding to the Logout Request**

1977 In step 4, the identity provider **MUST** respond to the <lib:LogoutRequest> with a SOAP 200
1978 OK <lib:LogoutResponse> message.

1979 **Step 5: Confirmation**

1980 In step 5, the user agent is sent an HTTP response that confirms the requested action of single
1981 logout was completed.

1982 **3.6 Identity Provider Introduction**

1983 This section defines the profiles by which a service provider discovers which identity providers a
1984 Principal is using. In identity federation networks having more than one identity provider, service
1985 providers need a means to discover which identity providers a Principal uses. The introduction
1986 profile relies on a cookie that is written in a domain that is common between identity providers and
1987 service providers in an identity federation network. The domain that the identity federation
1988 network predetermines for a deployment is known as the *common domain* in this specification, and
1989 the cookie containing the list of identity providers is known as the *common domain cookie*.

1990 Implementation of this profile is **OPTIONAL**. Whether identity providers and service providers
1991 implement this profile is a policy and deployment issue outside the scope of this specification.
1992 Also, which entities host web servers in the common domain is a deployment issue and is outside
1993 the scope of this specification.

1994 **3.6.1 Common Domain Cookie**

1995 The name of the cookie **MUST** be `_liberty_idp`. The format of the cookie content **MUST** be a
1996 list of base64-encoded (see [RFC2045](#)) identity provider succinct IDs separated by a single white
1997 space character. The identity provider IDs **MUST** adhere to the creation rules as defined in 3.2.2.2.
1998 The identity provider ID is a metadata element, as described in 0 and defined in
1999 [\[LibertyProtSchema\]](#).

2000 The common domain cookie writing service **SHOULD** append the identity provider ID to the list.
2001 If the identity provider ID is already present in the list, it **MAY** remove and append it when
2002 authentication of the Principal occurs. The intent is that the most recently established identity
2003 provider session is the last one in the list.

2004 The cookie **MUST** be set with no `Path` prefix or a `Path` prefix of `"/"`. The `Domain` **MUST** be set
2005 to `".[common-domain]"` where `[common-domain]` is the *common domain* established within
2006 the identity federation network for use with the introduction protocol. The cookie **MUST** be
2007 marked as `Secure`.

2008 The cookies **MAY** be either session or persistent (see [RFC2109](#)), the implementation of which is
2009 a policy and deployment issue of the identity federation network.

2010 **3.6.2 Setting the Common Domain Cookie**

2011 After the identity provider authenticates a Principal, it MAY set the common domain cookie. The
2012 means by which the identity provider sets the cookie are implementation-specific so long as the
2013 cookie is successfully set with the parameters given above. One possible implementation strategy
2014 follows and should be considered non-normative. The identity provider may:

- 2015 • Have previously established a DNS and IP alias for itself in the common domain
- 2016 • Redirect the user agent to itself using the DNS alias using a URL specifying "https" as
2017 the URL scheme. The structure of the URL is private to the implementation and may
2018 include session information needed to identify the user-agent.
- 2019 • Set the cookie on the redirected user agent using the parameters specified above
- 2020 • Redirect the user agent back to itself, or, if appropriate, to the service provider.

2021 **3.6.3 Obtaining the Common Domain Cookie**

2022 When a service provider needs to discover which identity providers the Principal uses, it invokes a
2023 protocol exchange designed to present the common domain cookie to the service provider after it
2024 is read by an HTTP server in the common domain.

2025 If the HTTP server in the common domain is operated by the service provider, the service provider
2026 MAY redirect the user agent to an identity provider's intersite transfer service for an optimized
2027 single sign-on process.

2028 The specific means by which the service provider reads the cookie are implementation-specific so
2029 long as it is able to cause the user agent to present cookies that have been set with the parameters
2030 given in section 3.6.1. One possible implementation strategy is described as follows and should be
2031 considered non-normative. Additionally, it may be sub-optimal for some applications.

- 2032 • Have previously established a DNS and IP alias for itself in the common domain
- 2033 • Redirect the user agent to itself using the DNS alias using a URL specifying "https" as
2034 the URL scheme. The structure of the URL is private to the implementation and should
2035 encode session state information needed to identify the user-agent, since it will not present
2036 any session cookies that were set using the service provider's original DNS name. This
2037 redirection could be performed via an HTTP 302 response or via an image tag in an
2038 HTML page sent to the user agent.
- 2039 • Read the cookie sent by the user agent, if any. Using the session state information encoded
2040 in the request URL, store the information retrieved from the cookie.
- 2041 • Use the session state information to determine what resource the user agent was attempting
2042 to access when it was redirected, if necessary. Send the user agent an HTTP 302 response
2043 redirecting it to the original resource.

2044 **4 Security Considerations**

2045 **4.1 Introduction**

2046 This section describes security considerations associated with Liberty protocols for identity
2047 federation, single sign-on, federation termination, and single logout.

2048 Liberty protocols, schemas, bindings, and profiles inherit and use extensively the SAML protocols.
2049 Therefore, the security considerations published along with the SAML specification have direct
2050 relevance (see [SAMLCore], [SAMLBind], and [SAMLSec]). Throughout this section if, for any
2051 reason, a specific consideration or countermeasure does not apply or differs, notice of this fact is
2052 made; and a description of alternatives is supplied, where possible.

2053 **4.2 General Requirements**

2054 **4.2.1 Security of SSL and TLS**

2055 SSL and TLS utilize a suite of possible cipher suites. The security of the SSL or TLS session
2056 depends on the chosen cipher suite. An entity (that is, a user agent, service provider, or identity
2057 provider) that terminates an SSL or TLS connection needs to offer (or accept) suitable cipher
2058 suites during the handshake. The following list of TLS 1.0 cipher suites (or their SSL 3.0
2059 equivalent) is recommended.

- 2060 • TLS_RSA_WITH_RC4_128_SHA
- 2061 • TLS_RSA_WITH_3DES_EDE_CBC_SHA
- 2062 • TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

2063 The above list is not exhaustive. The recommended cipher suites are among the most commonly
2064 used. Note: New cipher suites are added as they are standardized and should be considered for
2065 inclusion if they have sufficiently strong security properties. For example, it is anticipated that the
2066 AES-based cipher suites being standardized in the IETF will be widely adopted and deployed.

2067 **4.2.2 Security Implementation**

2068 The suitable implementation of security protocols is necessary to maintain the security of a system,
2069 including

- 2070 • Secure random or pseudo-random number generation
- 2071 • Secure storage

2072 **4.3 Classification of Threats**

2073 **4.3.1 Threat Model**

2074 For an analysis of threat classifications, an Internet threat model has been used. In other words, the
2075 threat model assumes that intermediary and end-systems participating in Liberty protocol
2076 exchanges have not been compromised. However, where possible, the consequences and
2077 containment properties of a compromised system entity are described and countermeasures are
2078 suggested to bolster the security posture so that the exposure from a security breach is minimized.

2079 Given the nature of the Internet, the assumption is made that deployment is across the global
2080 Internet and, therefore, crosses multiple administrative boundaries. Thus, an assumption is also
2081 made that the adversary has the capacity to engage in both passive and active attacks (see 4.3.3).

2082 **4.3.2 Rogue and Spurious Entities**

2083 Attackers may be classified based on their capabilities and the roles that they play in launching
2084 attacks on a Liberty system as follows:

- 2085 • **Rogue Entities:** Entities that misuse their privileges. The rogue actors may be Principals,
2086 user agents, service providers, or identity providers. A rogue Principal is a legitimate
2087 participant who attempts to escalate its privileges or masquerade as another system
2088 Principal. A rogue user agent may, for instance, misuse the relationships between its
2089 associated Principals and an identity provider to launch certain attacks. Similarly, a rogue
2090 service provider may be able to exploit the relationship that it has either with a Principal or
2091 with an identity provider to launch certain attacks.
- 2092 • **Spurious Entities:** Entities that masquerade as a legitimate entity or are completely
2093 unknown to the system. The spurious actors include Principals, user agents (i.e., user
2094 agents without associated legitimate Liberty Principals), service providers, or identity
2095 providers. A spurious service provider may, for instance, pretend to be a service provider
2096 that has a legitimate relationship with an identity provider. Similarly, a spurious Principal
2097 may be one that pretends to be a legitimate Principal that has a relationship with either a
2098 service provider or an identity provider.

2099 **4.3.3 Active and Passive Attackers**

2100 Both rogue and spurious entities may launch active or passive attacks on the system. In a passive
2101 attack the attacker does not inject traffic or modify traffic in any way. Such an attacker usually
2102 passively monitors the traffic flow, and the information that is obtained in that flow may be used at
2103 a later time. An active attacker, on the other hand, is capable of modifying existing traffic as well
2104 as injecting new traffic into the system.

2105 **4.3.4 Scenarios**

2106 The following scenarios describe possible attacks:

- 2107 • **Collusion: The secret cooperation between two or more Liberty entities to launch an**
2108 **attack, for example,**
 - 2109 • Collusion between Principal and service provider
 - 2110 • Collusion between Principal and identity provider
 - 2111 • Collusion between identity provider and service provider
 - 2112 • Collusion among two or more Principals
 - 2113 • Collusion between two or more service providers
 - 2114 • Collusion between two or more identity providers
- 2115 • **Denial-of-Service Attacks:** The prevention of authorized access to a system resource or
2116 the delaying of system operations and functions.
- 2117 • **Man-in-the-Middle Attacks:** A form of active wiretapping attack in which the attacker
2118 intercepts and selectively modifies communicated data to masquerade as one or more of
2119 the entities involved in a communication association.

- 2120 • **Replay Attacks:** An attack in which a valid data transmission is maliciously or
2121 fraudulently repeated, either by the originator or by an adversary who intercepts the data
2122 and retransmits it, possibly as part of a masquerade attack.
- 2123 • **Session Hijacking:** A form of active wiretapping in which the attacker seizes control of a
2124 previously established communication association.

2125 **4.4 Threat Scenarios and Countermeasures**

2126 In this section, threats that may apply to all the Liberty profiles are considered first. Threats that
2127 are specific to individual profiles are then considered. In each discussion the threat is described as
2128 well as the countermeasures that exist in the profile or the additional countermeasures that may be
2129 implemented to mitigate the threat.

2130 **4.4.1 Common Threats for All Profiles**

2131 **Threat:** Request messages sent in cleartext

2132 **Description:** Most profile protocol exchanges do not mandate that all exchanges
2133 commence over a secure communication channel. This lack of transport security potentially
2134 exposes requests and responses to both passive and active attacks.

2135 One obvious manifestation is when the initial contact is not over a secure transport and the
2136 Liberty profile begins to exchange messages by carrying the request message back to the
2137 user agent in the location header of a redirect.

2138 Another such manifestation could be a request or response message which carries a URI
2139 that may be resolved on a subsequent exchange, for instance
2140 `lib:AuthnContextClassRef`. If this URI were to specify a less or insecure transport,
2141 then the exchange may be vulnerable to the types of attacks described above.

2142 **Countermeasure:** Ensure that points of entry to Liberty protocol exchanges utilize the
2143 `https` URL `<scheme>` and that all interactions for that profile consistently exchange
2144 messages over `https`.

2145 **Threat:** Malicious redirects into identity or service provider targets

2146 **Description:** A spurious entity could issue a redirect to a user agent so that the user agent
2147 would access a resource that disrupts single sign-on. For example, an attacker could
2148 redirect the user agent to a logout resource of a service provider causing the Principal to be
2149 logged out of all existing authentication sessions.

2150 **Countermeasure:** Access to resources that produce side effects could be specified with a
2151 transient qualifier that must correspond to the current authentication session. Alternatively,
2152 a confirmation dialog could be interposed that relies on a transient qualifier with similar
2153 semantics.

2154 **Threat:** Relay state tampering or fabrication

2155 **Description:** Some of the messages may carry a `<lib:RelayState>` element, which is
2156 recommended to be integrity-protected by the producer and optionally confidentiality-
2157 protected. If these practices are not followed, an adversary could trigger unwanted side
2158 effects. In addition, by not confidentiality-protecting the value of this element, a legitimate

2159 system entity could inadvertently expose information to the identity provider or a passive
2160 attacker.

2161 **Countermeasure:** Follow the recommended practice of confidentiality- and integrity-
2162 protecting the `<lib:RelayState>` data. Note: Because the value of this element is both
2163 produced and consumed by the same system entity, symmetric cryptographic primitives
2164 could be utilized.

2165 4.4.2 Single Sign-On and Federation

2166 4.4.2.1 Common Interactions for All Single Sign-On and Federation Profiles

2167 **Threat:** `<lib:AuthnRequest>` sent over insecure channel

2168 **Description:** It is recommended that the initial exchange to access the intersite transfer
2169 service be conducted over a TLS-secured transport. Not following this recommendation
2170 can expose the exchange to both passive and active attacks.

2171 **Countermeasure:** Deploy the intersite transfer service under an `https` scheme.

2172 **Threat:** Unsigned `<lib:AuthnRequest>` message

2173 **Description:** The signature element of an `<lib:AuthnRequest>` is optional and thus the
2174 absence of the signature could pose a threat to the identity provider or even the targeted
2175 service provider. For example, a spurious system entity could generate an unsigned
2176 `<lib:AuthnRequest>` and redirect the user agent to the identity provider. The identity
2177 provider must then consume resources.

2178 **Countermeasure:** Sign the `<lib:AuthnRequest>`. The IDP can also verify the identity
2179 of the Principal in the absence of a signed request.

2180 **Threat:** Replay of an authentication assertion

2181 **Description:** After obtaining a valid assertion from an identity provider, either legitimately
2182 or surreptitiously, the entity replays the assertion to the Service at a later time. A digital
2183 signature must cover the entire assertion, thus elements within the assertion cannot be
2184 corrupted without detection during the mandatory verification step. However, it is possible
2185 to fabricate an `<lib:AuthnResponse>` with the valid assertion.

2186 **Countermeasure:** The issuer should sign `<lib:AuthnResponse>` messages. Signing
2187 binds the `<samlp:IssueInstant>` of the response message to the assertion it contains.
2188 This binding accords the relying party the opportunity to temporally judge the response.
2189 Additionally, a valid signature over the response binds the `<samlp:InResponseTo>`
2190 element to the corresponding `<lib:AuthnRequest>`. (Specifying a short period that the
2191 authentication assertion can be relied upon will minimize, but not mitigate this threat.
2192 Binding the `<lib:AssertionId>` to the request/`<samlp:InResponseTo>` element may
2193 also be handy.)

2194 **Threat:** Fabricated `<lib:AuthnResponse>` denial of service

2195 **Description:** An attacker captures the `<samlp:RequestID>` sent in an
2196 `<lib:AuthnRequest>` message by a service provider to an identity provider, and sends
2197 several spurious `<lib:AuthnResponse>` messages to the service provider with the same
2198 `<samlp:InResponseTo>`. Because the `<samlp:InResponseTo>` matches a

2199 <samlp:RequestID> that the service provider had used, the service provider goes
2200 through the process of validating the signature in the message. Thus, it is subject to a denial
2201 of service attack.

2202 **Countermeasure:** A secure communication channel should be established before
2203 transferring requests and responses.

2204 **Threat:** Collusion between two Principals

2205 **Description:** After getting an artifact or <lib:AuthnResponse> in step 6 (see 3.2.1), a
2206 legitimate Principal A could pass this artifact or <lib:AuthnResponse> on to another
2207 Principal, B. Principal B is now able to use the artifact or <lib:AuthnResponse>, while
2208 the actual authentication happened via Principal A.

2209 **Countermeasure:** Implementations where this threat is a concern MUST use the
2210 <saml:AuthenticationLocality> in the authentication statement. The IP address that
2211 Principal B uses would be different from the IP address within the
2212 <saml:AuthenticationLocality>. This countermeasure may not suffice when the
2213 user agent is behind a firewall or proxy server. IP spoofing may also circumvent this
2214 countermeasure.

2215 **Threat:** Stolen artifact and subsequent Principal impersonation

2216 **Description:** See Section 4.1.1.9.1 in [[SAMLBind](#)]

2217 **Countermeasure:** Identity providers MUST enforce a policy of one-time retrieval of the
2218 assertion corresponding to an artifact so that a stolen artifact can be used only once.
2219 Implementations where this threat is a concern MUST use the
2220 <saml:AuthenticationLocality> in the authentication statement. The IP address of a
2221 spurious user agent that attempts to use the stolen artifact would be different from IP
2222 address within the <saml:AuthenticationLocality>. The service provider may then
2223 be able to detect that the IP addresses differ. This countermeasure may not suffice when the
2224 user agent is behind a firewall or proxy server. IP address spoofing may also circumvent
2225 this countermeasure.

2226 **Threat:** Stolen assertion and subsequent Principal impersonation

2227 **Description:** See Section 4.1.1.9.1 in [[SAMLBind](#)]

2228 **Countermeasure:** Refer to the previous threat for requirements.

2229 **Threat:** Rogue service provider uses artifact or assertion to impersonate Principal at a different
2230 service provider

2231 **Description:** Because the <lib:AuthnResponse> contains the <lib:ProviderID>,
2232 this threat is not possible.

2233 **Countermeasure:** None

2234 **Threat:** Rogue identity provider impersonates Principal at a service provider

2235 **Description:** Because the Principal trusts the identity provider, it is assumed that the
2236 identity provider does not misuse the Principal's trust.

2237 **Countermeasure:** None

2238 **Threat:** Rogue user attempts to impersonate currently logged-in legitimate Principal and thereby
2239 gain access to protected resources.

2240 **Description:** Once a Principal is successfully logged into an identity provider, subsequent
2241 `<AuthnRequest>` messages from different service providers concerning that Principal
2242 will not necessarily cause the Principal to be reauthenticated. Principals must, however, be
2243 authenticated unless the identity provider can determine that an `<AuthnRequest>` is
2244 associated not only with the Principal's identity, but also with a validly authenticated
2245 identity provider session for that Principal.

2246 **Countermeasure:** In implementations where this threat is a concern, identity providers
2247 MUST maintain state information concerning active sessions, and MUST validate the
2248 correspondence between an `<AuthnRequest>` and an active session before issuing an
2249 `<AuthnResponse>` without first authenticating the Principal. Cookies posted by identity
2250 providers MAY be used to support this validation process, though Liberty does not
2251 mandate a cookie-based approach.

2252 4.4.2.2 Liberty-Enabled Client and Proxy Profile

2253 **Threat:** Intercepted `<lib:AuthnRequestEnvelope>` and `<lib:AuthnResponse>` and
2254 subsequent Principal impersonation.

2255 **Description:** A spurious system entity can interject itself as a man-in-the-middle (MITM)
2256 between the user agent (LECP) and a legitimate serviceprovider, where it acts in the
2257 service provider role in interactions with the (LECP), and in the user agent role in
2258 interactions with the legitimate service provider. In this way, as a first step, the MITM is
2259 able to intercept the service provider's `<lib:AuthnRequestEnvelope>` (step 3 of
2260 section 3.2.5) and substitute any URL of its choosing for the
2261 `<lib:AssertionConsumerServiceURL>` value before forwarding the
2262 `<lib:AuthnRequestEnvelope>` on to the LECP. Typically, the MITM will insert a
2263 URL value that points back to itself. Then, if the LECP subsequently receives a
2264 `<lib:AuthnResponseEnvelope>` from the identity provider (step 6 in section 3.2.5)
2265 and subsequently sends the contained `<lib:AuthnResponse>` to the
2266 `<lib:AssertionConsumerServiceURL>` received from the MITM, the MITM will be
2267 able to masquerade as the Principal at the legitimate service provider.

2268 **Countermeasure:** The identity provider specifies to the LECP the address to which the
2269 LECP must send the `<lib:AuthnResponse>`. The
2270 `<lib:AssertionConsumerServiceURL>` in the `<lib:AuthnResponseEnvelope>`
2271 element is for this purpose. This URL value is among the metadata that identity and service
2272 providers must exchange in the process of establishing their operational relationship (see
2273 sections 3.1 and 3.1.3).

2274 4.4.2.3 Federation

2275 **Threat:** Collusion among service providers can violate privacy of the Principal

2276 **Description:** When a group of service providers collude to share the
2277 `<lib:IDPProvidedNameIdentifier>` of a Principal, they can track and in general
2278 compromise the privacy of the principal. More generally, this threat exists for any common
2279 data (e.g. phone number) shared by rogue system entities.

2280 **Countermeasure:** The `<lib:IDPProvidedNameIdentifier>` is required to be unique
2281 for each identity provider to service provider relationship. However, this requirement does

2282 not eliminate the threat when there are rogue participants under the Principal's identity
2283 federation. The only protection is for Principals to be cautious when they choose service
2284 providers and understand their privacy policies.

2285 **Threat:** Poorly generated name identifiers may compromise privacy

2286 **Description:** The federation protocol mandates that the `<lib:NameIdentifier>`
2287 elements be unique within a Principal's federated identities. The name identifiers
2288 exchanged are pseudonyms and, to maintain the privacy of the Principal, should be
2289 resistant to guessing or derivation attacks.

2290 **Countermeasure:** Name identifiers should be constructed using pseudo-random values
2291 that have no discernable correspondence with the Principal's identifier (or name) used by
2292 the entity that generates the name identifier.

2293 **4.4.3 Name Registration**

2294 No known threats.

2295 **4.4.4 Federation Termination: HTTP-Redirect-Based Profile**

2296 **Threat:** Attacker can monitor and disrupt termination

2297 **Description:** During the initial steps, a passive attacker can collect the
2298 `<lib:FederationTerminationNotification>` information when it is issued in the
2299 redirect. This threat is possible because the first and second steps are not required to use
2300 https as the URL scheme. An active attacker may be able to intercept and modify the
2301 message conveyed in step 2 because the digital signature only covers a portion of the
2302 message. This initial exchange also exposes the name identifier. Exposing these data poses
2303 a privacy threat.

2304 **Countermeasure:** All exchanges should be conducted over a secure transport such as SSL
2305 or TLS.

2306 **4.4.5 Single Logout: HTTP-Redirect-Based Profile**

2307 **Threat:** Passive attacker can collect a Principal's name identifier

2308 **Description:** During the initial steps, a passive attacker can collect the
2309 `<lib:LogoutRequest>` information when it is issued in the redirect. Exposing these data
2310 poses a privacy threat.

2311 **Countermeasure:** All exchanges should be conducted over a secure transport such as SSL
2312 or TLS.

2313 **Threat:** Unsigned `<lib:LogoutRequest>` message

2314 **Description:** An Unsigned `<lib:LogoutRequest>` could be injected by a spurious
2315 system entity thus denying service to the Principal. Assuming that the `NameIdentifier`
2316 can be deduced or derived then it is conceivable that the user agent could be directed to
2317 deliver a fabricated `<lib:LogoutRequest>` message.

2318 **Countermeasure:** Sign the `<lib:LogoutRequest>` message. The identity provider can
2319 also verify the identity of a Principal in the absence of a signed request.

2320 **4.4.6 Identity Provider Introduction**

2321 No known threats.

2322 **5 References**

2323 **5.1 Normative**

- 2324 [HTML4] Raggett, D., Le Hors, A., & Jacobs, I., eds (December 1999). "HTML 4.01
2325 Specification," W3C Recommendation. World Wide Web Consortium.
2326 <<http://www.w3.org/TR/html401>> [18 December 2002].
- 2327 [LibertyGloss] Mauldin, H., & Wason, T., eds. (January 2003). "Liberty Architecture
2328 Glossary," Version 1.1. Liberty Alliance Project,
2329 <<http://www.projectliberty.org/specs/>>.
- 2330 [LibertyProtSchema] Beatty, J., & Kemp, J., eds. (January 2003). "Liberty Protocols and Schema
2331 Specification," Version 1.1. Liberty Alliance Project,
2332 <<http://www.projectliberty.org/specs/>>.
- 2333 [RFC1750] Eastlake 3rd, D., Croker, S., & Schiller, J. (December 1994). "Randomness
2334 Recommendations for Security," RFC 1750. The Internet Engineering Task
2335 Force, <<http://www.rfc-editor.org/rfc/rfc1750.txt>> [18 December 2002].
- 2336 [RFC1945] Berners-Lee, T., Fielding, R., Frystyk, H. (May 1996). "Hypertext Transfer
2337 Protocol -- HTTP/1.0," RFC 1945. The Internet Engineering Task Force,
2338 <<http://www.rfc-editor.org/rfc/rfc1945.txt>> [18 December 2002].
- 2339 [RFC2045] Freed, N., Borenstein, N. (November 1996). "Multipurpose Internet Mail
2340 Extensions (MIME) Part One: Format of Internet Message Bodies," RFC
2341 2045. The Internet Engineering Task Force, <[http://www.rfc-
2342 editor.org/rfc/rfc2045.txt](http://www.rfc-editor.org/rfc/rfc2045.txt)> [18 December 2002].
- 2343 [RFC2109] Kristol, D., & Montulli, L. (February 1997). "HTTP State Management
2344 Mechanism," RFC 2109. The Internet Engineering Task Force,
2345 <<http://www.rfc-editor.org/rfc/rfc2109.txt>> [18 December 2002].
- 2346 [RFC2119] Bradner, S. (March 1997). "Key words for use in RFCs to Indicate
2347 Requirement Levels," RFC 2119. The Internet Engineering Task Force,
2348 <<http://www.rfc-editor.org/rfc/rfc2119.txt>> [18 December 2002].
- 2349 [RFC2246] Dierks, T.,& Allen, C. (January 1999). "The TLS Protocol Version 1.0,"
2350 RFC 2246. The Internet Engineering Task Force, <[http://www.rfc-
2351 editor.org/rfc/rfc2246.txt](http://www.rfc-editor.org/rfc/rfc2246.txt)> [18 December 2002]
- 2352 [RFC2396] Berners-Lee, T., Fielding, R., & Masinter, L. (August 1998). "Uniform
2353 Resource Identifiers (URI): Generic Syntax," RFC 2396. The Internet
2354 Engineering Task Force, <<http://www.rfc-editor.org/rfc/rfc2396.txt>> [18
2355 December 2002].
- 2356 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., &
2357 Berners-Lee, T. (June 1999). "Hypertext Transfer Protocol -- HTTP/1.1,"

- 2358 RFC 2616. The Internet Engineering Task Force, <[http://www.rfc-](http://www.rfc-editor.org/rfc/rfc2616.txt)
2359 [editor.org/rfc/rfc2616.txt](http://www.rfc-editor.org/rfc/rfc2616.txt)> [18 December 2002].
- 2360 [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P.,
2361 Luotonen, A., & Stewart, L. (June 1999). "HTTP Authentication: Basic and
2362 Digest Access Authentication," RFC 2617. The Internet Engineering Task
2363 Force, <<http://www.rfc-editor.org/rfc/rfc2617.txt>> [18 December 2002]
- 2364 [RFC2774] Nielsen, H., Leach P., & Lawrence, S. (February 2000). "An HTTP
2365 Extension Framework," RFC 2774. The Internet Engineering Task Force,
2366 <<http://www.rfc-editor.org/rfc/rfc2774.txt>> [18 December 2002].
- 2367 [RFC3106] Eastlake, D., & Goldstein, T. (April 2001). "ECML v1.1: Field
2368 Specifications for E-Commerce," RFC 3106. The Internet Engineering Task
2369 Force, <<http://www.rfc-editor.org/rfc/rfc3106.txt>> [18 December 2002].
- 2370 [SAMLBind] Mishra, P., ed. (05 Nov. 2002). "Bindings and Profiles for the OASIS
2371 Security Assertion Markup Language (SAML)," Version 1.0, OASIS
2372 Standard. Organization for the Advancement of Structured Information
2373 Standards, <<http://www.oasis-open.org/committees/security/#documents>>
2374 [18 December 2002].
- 2375 [SAMLCore] Hallam-Baker, P., Maler, E., eds. (05 Nov. 2002). "Assertions and Protocol
2376 for the OASIS Security Assertion Markup Language (SAML)," Version 1.0,
2377 OASIS Standard. Organization for the Advancement of Structured
2378 Information Standards, <[http://www.oasis-](http://www.oasis-open.org/committees/security/#documents)
2379 [open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)> [18 December 2002].
- 2380 [SAMLGloss] Hodges, J., Maler, E, eds. (05 Nov. 2002). "Glossary for the OASIS
2381 Security Assertion Markup Language (SAML)," Version 1.0, OASIS
2382 Standard. Organization for the Advancement of Structured Information
2383 Standards, <<http://www.oasis-open.org/committees/security/#documents>>
2384 [18 December 2002].
- 2385 [SAMLReqs] Platt, D., Prodromou, E., eds. (05 Nov. 2002). "SAML Requirements and
2386 Use Cases," Version 1.0, OASIS Standard. Organization for the
2387 Advancement of Structured Information Standards, <[http://www.oasis-](http://www.oasis-open.org/committees/security/#documents)
2388 [open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)> [18 December 2002].
- 2389 [SAMLSec] McLaren, C., ed. (05 Nov. 2002). "Security Considerations for the OASIS
2390 Security Assertion Markup Language (SAML)," Version 1.0, OASIS
2391 Standard. Organization for the Advancement of Structured Information
2392 Standards, <<http://www.oasis-open.org/committees/security/#documents>>
2393 [18 December 2002].
- 2394 [Schema1] Thompson, H. S., Beech, D., Maloney, M., & Mendleson, N., eds. (May
2395 2002). "XML Schema Part 1: Structures," Recommendation. World Wide
2396 Web Consortium, <<http://www.w3.org/TR/xmlschema-1/>> [18 December
2397 2002].

- 2398 [Schema2] Biron, P. V., & Malhotra, A., eds. (May 2002). “XML Schema Part 2:
2399 Datatypes,” Recommendation. World Wide Web Consortium,
2400 <<http://www.w3.org/TR/xmlschema-2/>> [18 December 2002].
- 2401 [SOAP1.1] D. Box et al. (May 2000). “Simple Object Access Protocol (SOAP) 1.1,”
2402 Note. World Wide Web Consortium, <<http://www.w3.org/TR/SOAP>> [18
2403 December 2002].
- 2404 [SSLv3] Freier, A. O., Karlton, P., & Kocher, P. (November 1996). “The SSL
2405 Protocol,” Version 3.0, Internet Draft 02. Internet Engineering Task Force,
2406 <<http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt>> [18
2407 December 2002].
- 2408 [WML1.3] (February 2000). “Wireless Application Protocol Wireless Markup
2409 Language Specification” Version 1.3. Wireless Application Protocol Forum,
2410 Ltd., <<http://www.wapforum.org/what/technical.htm>> [18 December 2002].
- 2411 [XMLSig] Eastlake, D., Reagle, J., Solo, D., et al. (February 2002). “XML-Signature
2412 Syntax and Processing,” Recommendation. World Wide Web Consortium,
2413 <<http://www.w3.org/TR/xmlsig-core/>> [18 December 2002].
- 2414 **5.2 Informative**
- 2415 [Anders] Rundgren, A. (August 2001). “A suggestion on how to implement SAML
2416 browser bindings without using Artifacts.” <[http://www.x-
2417 obi.com/OBI400/andersr-browser-artifact.ppt](http://www.x-obi.com/OBI400/andersr-browser-artifact.ppt)> [18 December 2002].
- 2418 [Rescorla-Sec] Rescorla, E., & Korver, B., eds. (October 2002). “Guidelines for Writing
2419 RFC Text on Security Considerations,” Draft 01. The Internet Engineering
2420 Task Force. <<http://www.ietf.org/internet-drafts/draft-iab-sec-cons-01.txt>>
2421 [18 December 2002].
- 2422 [ShibMarlena] Erdos, M. (June 2001). “Shibboleth Architecture” Draft V1.1. Internet2,
2423 <[http://shibboleth.internet2.edu/docs/draft-erdos-shibboleth-architecture-
2424 01.pdf](http://shibboleth.internet2.edu/docs/draft-erdos-shibboleth-architecture-01.pdf)> [18 December 2002].
- 2425