



Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0

OASIS Standard, 15 March 2005

Document identifier:

saml-bindings-2.0-os

Location:

<http://docs.oasis-open.org/security/saml/v2.0/>

Editors:

Scott Cantor, Internet2
Frederick Hirsch, Nokia
John Kemp, Nokia
Rob Philpott, RSA Security
Eve Maler, Sun Microsystems

SAML V2.0 Contributors:

Conor P. Cahill, AOL
John Hughes, Atos Origin
Hal Lockhart, BEA Systems
Michael Beach, Boeing
Rebekah Metz, Booz Allen Hamilton
Rick Randall, Booz Allen Hamilton
Thomas Wisniewski, Entrust
Irving Reid, Hewlett-Packard
Paula Austel, IBM
Maryann Hondo, IBM
Michael McIntosh, IBM
Tony Nadalin, IBM
Nick Ragouzis, Individual
Scott Cantor, Internet2
RL 'Bob' Morgan, Internet2
Peter C Davis, Neustar
Jeff Hodges, Neustar
Frederick Hirsch, Nokia
John Kemp, Nokia
Paul Madsen, NTT
Steve Anderson, OpenNetwork
Prateek Mishra, Principal Identity
John Linn, RSA Security
Rob Philpott, RSA Security
Jahan Moreh, Sigaba
Anne Anderson, Sun Microsystems
Eve Maler, Sun Microsystems
Ron Monzillo, Sun Microsystems

45 Greg Whitehead, Trustgenix

46 **Abstract:**

47 This specification defines protocol bindings for the use of SAML assertions and request-response
48 messages in communications protocols and frameworks.

49 **Status:**

50 This is an **OASIS Standard** document produced by the Security Services Technical Committee. It
51 was approved by the OASIS membership on 1 March 2005.

52 Committee members should submit comments and potential errata to the [security-](mailto:security-services@lists.oasis-open.org)
53 [services@lists.oasis-open.org](mailto:security-services@lists.oasis-open.org) list. Others should submit them by filling out the web form located
54 at http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security. The
55 committee will publish on its web page (<http://www.oasis-open.org/committees/security>) a catalog
56 of any changes made to this document as a result of comments.

57 For information on whether any patents have been disclosed that may be essential to
58 implementing this specification, and any offers of patent licensing terms, please refer to the
59 Intellectual Property Rights web page for the Security Services TC ([http://www.oasis-](http://www.oasis-open.org/committees/security/ipr.php)
60 [open.org/committees/security/ipr.php](http://www.oasis-open.org/committees/security/ipr.php)).

Table of Contents

62	1 Introduction.....	5
63	1.1 Protocol Binding Concepts.....	5
64	1.2 Notation.....	5
65	2 Guidelines for Specifying Additional Protocol Bindings.....	7
66	3 Protocol Bindings.....	8
67	3.1 General Considerations.....	8
68	3.1.1 Use of RelayState.....	8
69	3.1.2 Security.....	8
70	3.1.2.1 Use of SSL 3.0 or TLS 1.0.....	8
71	3.1.2.2 Data Origin Authentication.....	8
72	3.1.2.3 Message Integrity.....	8
73	3.1.2.4 Message Confidentiality.....	9
74	3.1.2.5 Security Considerations.....	9
75	3.2 SAML SOAP Binding.....	9
76	3.2.1 Required Information.....	9
77	3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding.....	10
78	3.2.2.1 Basic Operation.....	10
79	3.2.2.2 SOAP Headers.....	10
80	3.2.3 Use of SOAP over HTTP.....	11
81	3.2.3.1 HTTP Headers.....	11
82	3.2.3.2 Caching.....	11
83	3.2.3.3 Error Reporting.....	11
84	3.2.3.4 Metadata Considerations.....	12
85	3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP.....	12
86	3.3 Reverse SOAP (PAOS) Binding.....	13
87	3.3.1 Required Information.....	13
88	3.3.2 Overview.....	13
89	3.3.3 Message Exchange.....	13
90	3.3.3.1 HTTP Request, SAML Request in SOAP Response.....	14
91	3.3.3.2 SAML Response in SOAP Request, HTTP Response.....	15
92	3.3.4 Caching.....	15
93	3.3.5 Security Considerations.....	15
94	3.3.5.1 Error Reporting.....	15
95	3.3.5.2 Metadata Considerations.....	15
96	3.4 HTTP Redirect Binding.....	15
97	3.4.1 Required Information.....	16
98	3.4.2 Overview.....	16
99	3.4.3 RelayState.....	16
100	3.4.4 Message Encoding.....	16
101	3.4.4.1 DEFLATE Encoding.....	17
102	3.4.5 Message Exchange.....	18
103	3.4.5.1 HTTP and Caching Considerations.....	19
104	3.4.5.2 Security Considerations.....	19
105	3.4.6 Error Reporting.....	20
106	3.4.7 Metadata Considerations.....	20
107	3.4.8 Example SAML Message Exchange Using HTTP Redirect.....	20

108	3.5 HTTP POST Binding.....	21
109	3.5.1 Required Information.....	21
110	3.5.2 Overview.....	21
111	3.5.3 RelayState.....	22
112	3.5.4 Message Encoding.....	22
113	3.5.5 Message Exchange.....	22
114	3.5.5.1 HTTP and Caching Considerations.....	23
115	3.5.5.2 Security Considerations.....	24
116	3.5.6 Error Reporting.....	24
117	3.5.7 Metadata Considerations.....	24
118	3.5.8 Example SAML Message Exchange Using HTTP POST.....	24
119	3.6 HTTP Artifact Binding.....	26
120	3.6.1 Required Information.....	26
121	3.6.2 Overview.....	27
122	3.6.3 Message Encoding.....	27
123	3.6.3.1 RelayState.....	27
124	3.6.3.2 URL Encoding.....	27
125	3.6.3.3 Form Encoding.....	28
126	3.6.4 Artifact Format.....	28
127	3.6.4.1 Required Information.....	28
128	3.6.4.2 Format Details.....	29
129	3.6.5 Message Exchange.....	29
130	3.6.5.1 HTTP and Caching Considerations.....	31
131	3.6.5.2 Security Considerations.....	31
132	3.6.6 Error Reporting.....	32
133	3.6.7 Metadata Considerations.....	32
134	3.6.8 Example SAML Message Exchange Using HTTP Artifact.....	32
135	3.7 SAML URI Binding.....	35
136	3.7.1 Required Information.....	35
137	3.7.2 Protocol-Independent Aspects of the SAML URI Binding.....	35
138	3.7.2.1 Basic Operation.....	35
139	3.7.3 Security Considerations.....	36
140	3.7.4 MIME Encapsulation.....	36
141	3.7.5 Use of HTTP URIs.....	36
142	3.7.5.1 URI Syntax.....	36
143	3.7.5.2 HTTP and Caching Considerations.....	36
144	3.7.5.3 Security Considerations.....	36
145	3.7.5.4 Error Reporting.....	37
146	3.7.5.5 Metadata Considerations.....	37
147	3.7.5.6 Example SAML Message Exchange Using an HTTP URI.....	37
148	4 References.....	38
149	Appendix A. Registration of MIME media type application/samlassertion+xml.....	40
150	Appendix B. Acknowledgments.....	44
151	Appendix C. Notices.....	46

1 Introduction

152

153 This document specifies SAML protocol bindings for the use of SAML assertions and request-response
154 messages in communications protocols and frameworks.

155 The SAML assertions and protocols specification [SAMLCore] defines the SAML assertions and request-
156 response messages themselves, and the SAML profiles specification [SAMLProfile] defines specific
157 usage patterns that reference both [SAMLCore] and bindings defined in this specification or elsewhere.
158 The SAML conformance document [SAMLConform] lists all of the specifications that comprise SAML
159 V2.0.

1.1 Protocol Binding Concepts

160

161 Mappings of SAML request-response message exchanges onto standard messaging or communication
162 protocols are called SAML *protocol bindings* (or just *bindings*). An instance of mapping SAML request-
163 response message exchanges into a specific communication protocol <FOO> is termed a <FOO> *binding*
164 *for SAML* or a *SAML <FOO> binding*.

165 For example, a SAML SOAP binding describes how SAML request and response message exchanges
166 are mapped into SOAP message exchanges.

167 The intent of this specification is to specify a selected set of bindings in sufficient detail to ensure that
168 independently implemented SAML-conforming software can interoperate when using standard messaging
169 or communication protocols.

170 Unless otherwise specified, a binding should be understood to support the transmission of any SAML
171 protocol message derived from the **samlp:RequestAbstractType** and **samlp:StatusResponseType**
172 types. Further, when a binding refers to "SAML requests and responses", it should be understood to mean
173 any protocol messages derived from those types.

174 For other terms and concepts that are specific to SAML, refer to the SAML glossary [SAMLGloss].

1.2 Notation

175

176 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
177 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as
178 described in IETF RFC 2119 [RFC2119].

179 `Listings of productions or other normative code appear like this.`

180 `Example code listings appear like this.`

181 **Note:** Notes like this are sometimes used to highlight non-normative commentary.

182 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
183 namespaces as follows, whether or not a namespace declaration is present in the example:

Prefix	XML Namespace	Comments
saml:	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace [SAMLCore].
samlp:	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace [SAMLCore].

Prefix	XML Namespace	Comments
ds:	http://www.w3.org/2000/09/xmldsig#	This namespace is defined in the XML Signature Syntax and Processing specification [XMLSig] and its governing schema.
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This namespace is defined in SOAP V1.1 [SOAP11].

184 This specification uses the following typographical conventions in text: <ns:Element>, XMLAttribute,
185 **Datatype**, OtherKeyword. In some cases, angle brackets are used to indicate non-terminals, rather than
186 XML elements; the intent will be clear from the context.

2 Guidelines for Specifying Additional Protocol Bindings

187

188

189 This specification defines a selected set of protocol bindings, but others will possibly be developed in the
190 future. It is not possible for the OASIS Security Services Technical Committee (SSTC) to standardize all of
191 these additional bindings for two reasons: it has limited resources and it does not own the standardization
192 process for all of the technologies used. This section offers guidelines for third parties who wish to specify
193 additional bindings.

194 The SSTC welcomes submission of proposals from OASIS members for new protocol bindings. OASIS
195 members may wish to submit these proposals for consideration by the SSTC in a future version of this
196 specification. Other members may simply wish to inform the committee of their work related to SAML.
197 Please refer to the SSTC web site [SSTCWeb] for further details on how to submit such proposals to the
198 SSTC.

199 Following is a checklist of issues that **MUST** be addressed by each protocol binding:

- 200 1. Specify three pieces of identifying information: a URI that uniquely identifies the protocol binding,
201 postal or electronic contact information for the author, and a reference to previously defined
202 bindings or profiles that the new binding updates or obsoletes.
- 203 2. Describe the set of interactions between parties involved in the binding. Any restrictions on
204 applications used by each party and the protocols involved in each interaction must be explicitly
205 called out.
- 206 3. Identify the parties involved in each interaction, including how many parties are involved and
207 whether intermediaries may be involved.
- 208 4. Specify the method of authentication of parties involved in each interaction, including whether
209 authentication is required and acceptable authentication types.
- 210 5. Identify the level of support for message integrity, including the mechanisms used to ensure
211 message integrity.
- 212 6. Identify the level of support for confidentiality, including whether a third party may view the contents
213 of SAML messages and assertions, whether the binding requires confidentiality, and the
214 mechanisms recommended for achieving confidentiality.
- 215 7. Identify the error states, including the error states at each participant, especially those that receive
216 and process SAML assertions or messages.
- 217 8. Identify security considerations, including analysis of threats and description of countermeasures.
- 218 9. Identify metadata considerations, such that support for a binding involving a particular
219 communications protocol or used in a particular profile can be advertised in an efficient and
220 interoperable way.

221 **3 Protocol Bindings**

222 The following sections define the protocol bindings that are specified as part of the SAML standard.

223 **3.1 General Considerations**

224 The following sections describe normative characteristics of all protocol bindings defined for SAML.

225 **3.1.1 Use of RelayState**

226 Some bindings define a "RelayState" mechanism for preserving and conveying state information. When
227 such a mechanism is used in conveying a request message as the initial step of a SAML protocol, it
228 places requirements on the selection and use of the binding subsequently used to convey the response.
229 Namely, if a SAML request message is accompanied by RelayState data, then the SAML responder
230 MUST return its SAML protocol response using a binding that also supports a RelayState mechanism, and
231 it MUST place the exact RelayState data it received with the request into the corresponding RelayState
232 parameter in the response.

233 **3.1.2 Security**

234 Unless stated otherwise, these security statements apply to all bindings. Bindings may also make
235 additional statements about these security features.

236 **3.1.2.1 Use of SSL 3.0 or TLS 1.0**

237 Unless otherwise specified, in any SAML binding's use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246], servers
238 MUST authenticate to clients using a X.509 v3 certificate. The client MUST establish server identity based
239 on contents of the certificate (typically through examination of the certificate's subject DN field,
240 subjectAltName attribute, etc.).

241 **3.1.2.2 Data Origin Authentication**

242 Authentication of both the SAML requester and the SAML responder associated with a message is
243 OPTIONAL and depends on the environment of use. Authentication mechanisms available at the SOAP
244 message exchange layer or from the underlying substrate protocol (for example in many bindings the
245 SSL/TLS or HTTP protocol) MAY be utilized to provide data origin authentication.

246 Transport authentication will not meet end-end origin-authentication requirements in bindings where the
247 SAML protocol message passes through an intermediary – in this case message authentication is
248 recommended.

249 Note that SAML itself offers mechanisms for parties to authenticate to one another, but in addition SAML
250 may use other authentication mechanisms to provide security for SAML itself.

251 **3.1.2.3 Message Integrity**

252 Message integrity of both SAML requests and SAML responses is OPTIONAL and depends on the
253 environment of use. The security layer in the underlying substrate protocol or a mechanism at the SOAP
254 message exchange layer MAY be used to ensure message integrity.

255 Transport integrity will not meet end-end integrity requirements in bindings where the SAML protocol
256 message passes through an intermediary – in this case message integrity is recommended.

257 **3.1.2.4 Message Confidentiality**

258 Message confidentiality of both SAML requests and SAML responses is OPTIONAL and depends on the
259 environment of use. The security layer in the underlying substrate protocol or a mechanism at the SOAP
260 message exchange layer MAY be used to ensure message confidentiality.

261 Transport confidentiality will not meet end-end confidentiality requirements in bindings where the SAML
262 protocol message passes through an intermediary.

263 **3.1.2.5 Security Considerations**

264 Before deployment, each combination of authentication, message integrity, and confidentiality
265 mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange and
266 the deployment environment. See specific protocol processing rules in [SAMLCore] and the SAML security
267 considerations document [SAMLSecure] for a detailed discussion.

268 IETF RFC 2617 [RFC2617] describes possible attacks in the HTTP environment when basic or message-
269 digest authentication schemes are used.

270 Special care should be given to the impact of possible caching on security.

271 **3.2 SAML SOAP Binding**

272 SOAP is a lightweight protocol intended for exchanging structured information in a decentralized,
273 distributed environment [SOAP11]. It uses XML technologies to define an extensible messaging
274 framework providing a message construct that can be exchanged over a variety of underlying protocols.
275 The framework has been designed to be independent of any particular programming model and other
276 implementation specific semantics. Two major design goals for SOAP are simplicity and extensibility.
277 SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often
278 found in distributed systems. Such features include but are not limited to "reliability", "security",
279 "correlation", "routing", and "Message Exchange Patterns" (MEPs).

280 A SOAP message is fundamentally a one-way transmission between SOAP nodes from a SOAP sender
281 to a SOAP receiver, possibly routed through one or more SOAP intermediaries. SOAP messages are
282 expected to be combined by applications to implement more complex interaction patterns ranging from
283 request/response to multiple, back-and-forth "conversational" exchanges [SOAP-PRIMER].

284 SOAP defines an XML message envelope that includes header and body sections, allowing data and
285 control information to be transmitted. SOAP also defines processing rules associated with this envelope
286 and an HTTP binding for SOAP message transmission.

287 The SAML SOAP binding defines how to use SOAP to send and receive SAML requests and responses.

288 Like SAML, SOAP can be used over multiple underlying transports. This binding has protocol-independent
289 aspects, but also calls out the use of SOAP over HTTP as REQUIRED (mandatory to implement).

290 **3.2.1 Required Information**

291 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:SOAP

292 **Contact information:** security-services-comment@lists.oasis-open.org

293 **Description:** Given below.

294 **Updates:** urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding

295 3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

296 The following sections define aspects of the SAML SOAP binding that are independent of the underlying
297 protocol, such as HTTP, on which the SOAP messages are transported. Note this binding only supports
298 the use of SOAP 1.1.

299 3.2.2.1 Basic Operation

300 SOAP 1.1 messages consist of three elements: an envelope, header data, and a message body. SAML
301 request-response protocol elements MUST be enclosed within the SOAP message body.

302 SOAP 1.1 also defines an optional data encoding system. This system is not used within the SAML SOAP
303 binding. This means that SAML messages can be transported using SOAP without re-encoding from the
304 "standard" SAML schema to one based on the SOAP encoding.

305 The system model used for SAML conversations over SOAP is a simple request-response model.

- 306 1. A system entity acting as a SAML requester transmits a SAML request element within the body of
307 a SOAP message to a system entity acting as a SAML responder. The SAML requester MUST
308 NOT include more than one SAML request per SOAP message or include any additional XML
309 elements in the SOAP body.
- 310 2. The SAML responder MUST return either a SAML response element within the body of another
311 SOAP message or generate a SOAP fault. The SAML responder MUST NOT include more than
312 one SAML response per SOAP message or include any additional XML elements in the SOAP
313 body. If a SAML responder cannot, for some reason, process a SAML request, it MUST generate a
314 SOAP fault. SOAP fault codes MUST NOT be sent for errors within the SAML problem domain, for
315 example, inability to find an extension schema or as a signal that the subject is not authorized to
316 access a resource in an authorization query. (SOAP 1.1 faults and fault codes are discussed in
317 [SOAP11] Section 4.1.)

318 On receiving a SAML response in a SOAP message, the SAML requester MUST NOT send a fault code
319 or other error messages to the SAML responder. Since the format for the message interchange is a
320 simple request-response pattern, adding additional items such as error conditions would needlessly
321 complicate the protocol.

322 [SOAP11] references an early draft of the XML Schema specification including an obsolete namespace.
323 SAML requesters SHOULD generate SOAP documents referencing only the final XML schema
324 namespace. SAML responders MUST be able to process both the XML schema namespace used in
325 [SOAP11] as well as the final XML schema namespace.

326 3.2.2.2 SOAP Headers

327 A SAML requester in a SAML conversation over SOAP MAY add arbitrary headers to the SOAP message.
328 This binding does not define any additional SOAP headers.

329 **Note:** The reason other headers need to be allowed is that some SOAP software and
330 libraries might add headers to a SOAP message that are out of the control of the SAML-
331 aware process. Also, some headers might be needed for underlying protocols that require
332 routing of messages or by message security mechanisms.

333 A SAML responder MUST NOT require any headers in the SOAP message in order to process the SAML
334 message correctly itself, but MAY require additional headers that address underlying routing or message
335 security requirements.

336 **Note:** The rationale is that requiring extra headers will cause fragmentation of the SAML
337 standard and will hurt interoperability.

338 3.2.3 Use of SOAP over HTTP

339 A SAML processor that claims conformance to the SAML SOAP binding MUST implement SAML over
340 SOAP over HTTP. This section describes certain specifics of using SOAP over HTTP, including HTTP
341 headers, caching, and error reporting.

342 The HTTP binding for SOAP is described in [SOAP11] Section 6.0. It requires the use of a `SOAPAction`
343 header as part of a SOAP HTTP request. A SAML responder MUST NOT depend on the value of this
344 header. A SAML requester MAY set the value of the `SOAPAction` header as follows:

345 `http://www.oasis-open.org/committees/security`

346 3.2.3.1 HTTP Headers

347 A SAML requester in a SAML conversation over SOAP over HTTP MAY add arbitrary headers to the
348 HTTP request. This binding does not define any additional HTTP headers.

349 **Note:** The reason other headers need to be allowed is that some HTTP software and
350 libraries might add headers to an HTTP message that are out of the control of the SAML-
351 aware process. Also, some headers might be needed for underlying protocols that require
352 routing of messages or by message security mechanisms.

353 A SAML responder MUST NOT require any headers in the HTTP request to correctly process the SAML
354 message itself, but MAY require additional headers that address underlying routing or message security
355 requirements.

356 **Note:** The rationale is that requiring extra headers will cause fragmentation of the SAML
357 standard and will hurt interoperability.

358 3.2.3.2 Caching

359 HTTP proxies should not cache SAML protocol messages. To ensure this, the following rules SHOULD be
360 followed.

361 When using HTTP 1.1 [RFC2616], requesters SHOULD:

- 362 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 363 • Include a `Pragma` header field set to "no-cache".

364 When using HTTP 1.1, responders SHOULD:

- 365 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
366 private".
- 367 • Include a `Pragma` header field set to "no-cache".
- 368 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

369 3.2.3.3 Error Reporting

370 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
371 return a "403 Forbidden" response. In this case, the content of the HTTP body is not significant.

372 As described in [SOAP11] Section 6.2, in the case of a SOAP error while processing a SOAP request, the
373 SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP
374 message in the response with a SOAP `<SOAP-ENV:fault>` element. This type of error SHOULD be
375 returned for SOAP-related errors detected before control is passed to the SAML processor, or when the
376 SOAP processor reports an internal error (for example, the SOAP XML namespace is incorrect, the SAML
377 schema cannot be located, the SAML processor throws an exception, and so on).

378 In the case of a SAML processing error, the SOAP HTTP server MUST respond with "200 OK" and
379 include a SAML-specified <samlp:Status> element in the SAML response within the SOAP body. Note
380 that the <samlp:Status> element does not appear by itself in the SOAP body, but only within a SAML
381 response of some sort.

382 For more information about the use of SAML status codes, see the SAML assertions and protocols
383 specification [SAMLCore].

384 3.2.3.4 Metadata Considerations

385 Support for the SOAP binding SHOULD be reflected by indicating either a URL endpoint at which requests
386 contained in SOAP messages for a particular protocol or profile are to be sent, or alternatively with a
387 WSDL port/endpoint definition.

388 3.2.3.5 Example SAML Message Exchange Using SOAP over HTTP

389 Following is an example of a query that asks for an assertion containing an attribute statement from a
390 SAML attribute authority.

```
391 POST /SamlService HTTP/1.1
392 Host: www.example.com
393 Content-Type: text/xml
394 Content-Length: nnn
395 SOAPAction: http://www.oasis-open.org/committees/security
396 <SOAP-ENV:Envelope
397   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
398   <SOAP-ENV:Body>
399     <samlp:AttributeQuery xmlns:samlp:="..."
400     xmlns:saml="..." xmlns:ds="..." ID="_6c3a4f8b9c2d" Version="2.0"
401     IssueInstant="2004-03-27T08:41:00Z"
402       <ds:Signature> ... </ds:Signature>
403       <saml:Subject>
404         ...
405       </saml:Subject>
406     </samlp:AttributeQuery>
407   </SOAP-ENV:Body>
408 </SOAP-ENV:Envelope>
```

409 Following is an example of the corresponding response, which supplies an assertion containing the
410 attribute statement as requested.

```
411 HTTP/1.1 200 OK
412 Content-Type: text/xml
413 Content-Length: nnnn
414 <SOAP-ENV:Envelope
415   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
416   <SOAP-ENV:Body>
417     <samlp:Response xmlns:samlp:="..." xmlns:saml="..." xmlns:ds="..."
418     ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:00Z">
419       <saml:Issuer>https://www.example.com/SAML</saml:Issuer>
420       <ds:Signature> ... </ds:Signature>
421       <Status>
422         <StatusCode Value="..." />
423       </Status>
424
425       <saml:Assertion>
426         <saml:Subject>
427           ...
428         </saml:Subject>
429         <saml:AttributeStatement>
430           ...
431         </saml:AttributeStatement>
432       </saml:Assertion>
433     </samlp:Response>
434   </SOAP-Env:Body>
```

436 3.3 Reverse SOAP (PAOS) Binding

437 This binding leverages the Reverse HTTP Binding for SOAP specification [PAOS]. Implementers MUST
438 comply with the general processing rules specified in [PAOS] in addition to those specified in this
439 document. In case of conflict, [PAOS] is normative.

440 3.3.1 Required Information

441 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:PAOS

442 **Contact information:** security-services-comment@lists.oasis-open.org

443 **Description:** Given below.

444 **Updates:** None.

445 3.3.2 Overview

446 The reverse SOAP binding is a mechanism by which an HTTP requester can advertise the ability to act as
447 a SOAP responder or a SOAP intermediary to a SAML requester. The HTTP requester is able to support
448 a pattern where a SAML request is sent to it in a SOAP envelope in an HTTP response from the SAML
449 requester, and the HTTP requester responds with a SAML response in a SOAP envelope in a subsequent
450 HTTP request. This message exchange pattern supports the use case defined in the ECP SSO profile
451 (described in the SAML profiles specification [SAMLProfile]), in which the HTTP requester is an
452 intermediary in an authentication exchange.

453 3.3.3 Message Exchange

454 The PAOS binding includes two component message exchange patterns:

- 455 1. The HTTP requester sends an HTTP request to a SAML requester. The SAML requester responds
456 with an HTTP response containing a SOAP envelope containing a SAML request message.
- 457 2. Subsequently, the HTTP requester sends an HTTP request to the original SAML requester
458 containing a SOAP envelope containing a SAML response message. The SAML requester
459 responds with an HTTP response, possibly in response to the original service request in step 1.

460 The ECP profile uses the PAOS binding to provide authentication of the client to the service provider
461 before the service is provided. This occurs in the following steps, illustrated in Figure A:

- 462 1. The client requests a service using an HTTP request.
- 463 2. The service provider responds with a SAML authentication request. This is sent using a SOAP
464 request, carried in the HTTP response.
- 465 3. The client returns a SOAP response carrying a SAML authentication response. This is sent using a
466 new HTTP request.
- 467 4. Assuming the service provider authentication and authorization is successful, the service provider
468 may respond to the original service request in the HTTP response.

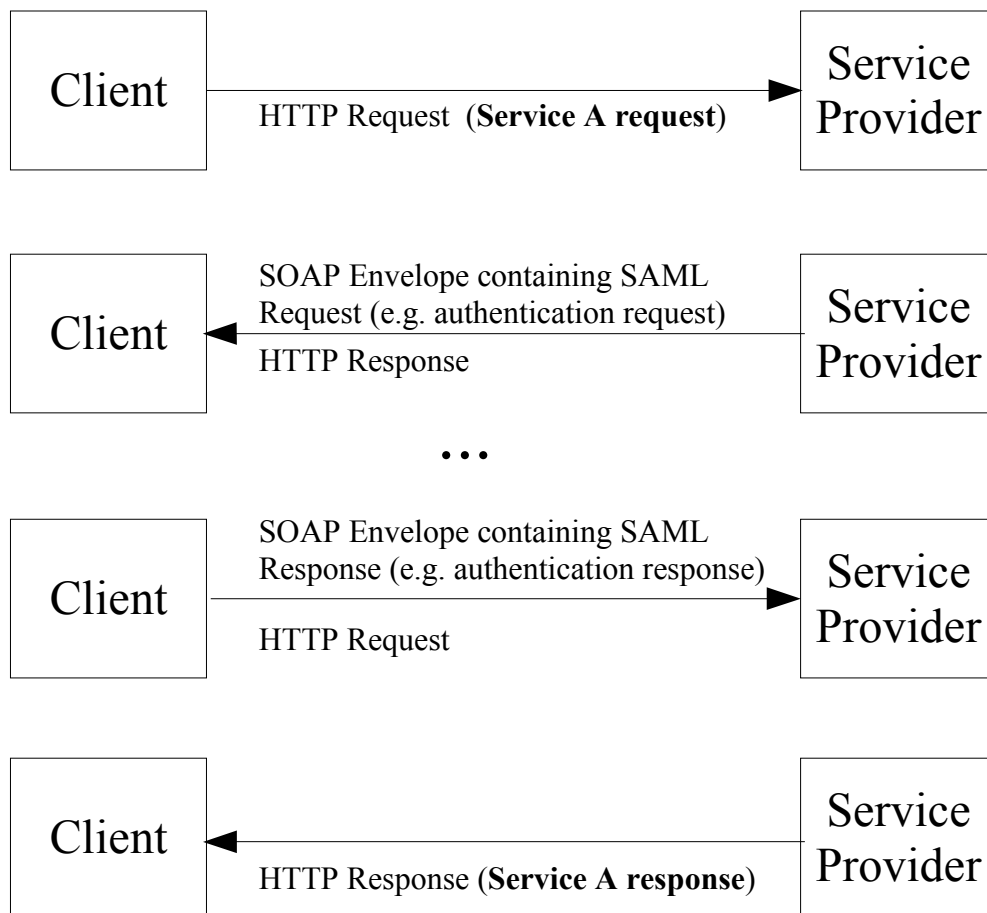


Figure 1: PAOS Binding Message Exchanges

469 The HTTP requester advertises the ability to handle this reverse SOAP binding in its HTTP requests using
 470 the HTTP headers defined by the PAOS specification. Specifically:

- 471 • The HTTP `Accept` Header field MUST indicate an ability to accept the
- 472 "application/vnd.paos+xml" content type.
- 473 • The HTTP `PAOS` Header field MUST be present and specify the PAOS version with
- 474 "urn:liberty:paos:2003-08" at a minimum.

475 Additional PAOS headers such as the service value MAY be specified by profiles that use the PAOS
 476 binding. The HTTP requester MAY add arbitrary headers to the HTTP request.

477 Note that this binding does not define a RelayState mechanism. Specific profiles that make use of this
 478 binding must therefore define such a mechanism, if needed. The use of a SOAP header is suggested for
 479 this purpose.

480 The following sections provide more detail on the two steps of the message exchange.

481 3.3.3.1 HTTP Request, SAML Request in SOAP Response

482 In response to an arbitrary HTTP request, the HTTP responder MAY return a SAML request message
 483 using this binding by returning a SOAP 1.1 envelope in the HTTP response containing a single SAML
 484 request message in the SOAP body, with no additional body content. The SOAP envelope MAY contain
 485 arbitrary SOAP headers defined by PAOS, SAML profiles, or additional specifications.

486 Note that while the SAML request message is delivered to the HTTP requester, the actual intended

487 recipient MAY be another system entity, with the HTTP requester acting as an intermediary, as defined by
488 specific profiles.

489 **3.3.3.2 SAML Response in SOAP Request, HTTP Response**

490 When the HTTP requester delivers a SAML response message to the intended recipient using the PAOS
491 binding, it places it as the only element in the SOAP body in a SOAP envelope in an HTTP request. The
492 HTTP requester may or may not be the originator of the SAML response. The SOAP envelope MAY
493 contain arbitrary SOAP headers defined by PAOS, SAML profiles, or additional specifications. The SAML
494 exchange is considered complete and the HTTP response is unspecified by this binding.

495 Profiles MAY define additional constraints on the HTTP content of non-SOAP responses during the
496 exchanges covered by this binding.

497 **3.3.4 Caching**

498 HTTP proxies should not cache SAML protocol messages. To ensure this, the following rules SHOULD be
499 followed.

500 When using HTTP 1.1, requesters sending SAML protocol messages SHOULD:

- 501 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 502 • Include a `Pragma` header field set to "no-cache".

503 When using HTTP 1.1, responders returning SAML protocol messages SHOULD:

- 504 • Include a `Cache-Control` header field set to "no-cache, no-store, must-revalidate,
505 private".
- 506 • Include a `Pragma` header field set to "no-cache".
- 507 • NOT include a `Validator`, such as a `Last-Modified` or `ETag` header.

508 **3.3.5 Security Considerations**

509 The HTTP requester in the PAOS binding may act as a SOAP intermediary and when it does, transport
510 layer security for origin authentication, integrity and confidentiality may not meet end-end security
511 requirements. In this case security at the SOAP message layer is recommended.

512 **3.3.5.1 Error Reporting**

513 Standard HTTP and SOAP error conventions MUST be observed. Errors that occur during SAML
514 processing MUST NOT be signaled at the HTTP or SOAP layer and MUST be handled using SAML
515 response messages with an error `<samlp:Status>` element.

516 **3.3.5.2 Metadata Considerations**

517 Support for the PAOS binding SHOULD be reflected by indicating a URL endpoint at which HTTP
518 requests and/or SAML protocol messages contained in SOAP envelopes for a particular protocol or profile
519 are to be sent. Either a single endpoint or distinct request and response endpoints MAY be supplied.

520 **3.4 HTTP Redirect Binding**

521 The HTTP Redirect binding defines a mechanism by which SAML protocol messages can be transmitted
522 within URL parameters. Permissible URL length is theoretically infinite, but unpredictably limited in
523 practice. Therefore, specialized encodings are needed to carry XML messages on a URL, and larger or

524 more complex message content can be sent using the HTTP POST or Artifact bindings.

525 This binding MAY be composed with the HTTP POST binding (see Section 3.5) and the HTTP Artifact
526 binding (see Section 3.6) to transmit request and response messages in a single protocol exchange using
527 two different bindings.

528 This binding involves the use of a message encoding. While the definition of this binding includes the
529 definition of one particular message encoding, others MAY be defined and used.

530 3.4.1 Required Information

531 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect

532 **Contact information:** security-services-comment@lists.oasis-open.org

533 **Description:** Given below.

534 **Updates:** None.

535 3.4.2 Overview

536 The HTTP Redirect binding is intended for cases in which the SAML requester and responder need to
537 communicate using an HTTP user agent (as defined in HTTP 1.1 [RFC2616]) as an intermediary. This
538 may be necessary, for example, if the communicating parties do not share a direct path of communication.
539 It may also be needed if the responder requires an interaction with the user agent in order to fulfill the
540 request, such as when the user agent must authenticate to it.

541 Note that some HTTP user agents may have the capacity to play a more active role in the protocol
542 exchange and may support other bindings that use HTTP, such as the SOAP and Reverse SOAP
543 bindings. This binding assumes nothing apart from the capabilities of a common web browser.

544 3.4.3 RelayState

545 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
546 MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the
547 message independent of any other protections that may or may not exist during message transmission.
548 Signing is not realistic given the space limitation, but because the value is exposed to third-party
549 tampering, the entity SHOULD ensure that the value has not been tampered with by using a checksum, a
550 pseudo-random value, or similar means.

551 If a SAML request message is accompanied by RelayState data, then the SAML responder MUST return
552 its SAML protocol response using a binding that also supports a RelayState mechanism, and it MUST
553 place the exact data it received with the request into the corresponding RelayState parameter in the
554 response.

555 If no such value is included with a SAML request message, or if the SAML response message is being
556 generated without a corresponding request, then the SAML responder MAY include RelayState data to be
557 interpreted by the recipient based on the use of a profile or prior agreement between the parties.

558 3.4.4 Message Encoding

559 Messages are encoded for use with this binding using a URL encoding technique, and transmitted using
560 the HTTP GET method. There are many possible ways to encode XML into a URL, depending on the
561 constraints in effect. This specification defines one such method without precluding others. Binding
562 endpoints SHOULD indicate which encodings they support using metadata, when appropriate. Particular
563 encodings MUST be uniquely identified with a URI when defined. It is not a requirement that all possible
564 SAML messages be encodable with a particular set of rules, but the rules MUST clearly indicate which
565 messages or content can or cannot be so encoded.

566 A URL encoding MUST place the message entirely within the URL query string, and MUST reserve the
567 rest of the URL for the endpoint of the message recipient.

568 A query string parameter named `SAMLEncoding` is reserved to identify the encoding mechanism used. If
569 this parameter is omitted, then the value is assumed to be
570 `urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE`.

571 All endpoints that support this binding MUST support the DEFLATE encoding described in the following
572 sub-section.

573 3.4.4.1 DEFLATE Encoding

574 **Identification:** `urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE`

575 SAML protocol messages can be encoded into a URL via the DEFLATE compression method (see
576 [RFC1951]). In such an encoding, the following procedure should be applied to the original SAML protocol
577 message's XML serialization:

- 578 1. Any signature on the SAML protocol message, including the `<ds:Signature>` XML element itself,
579 MUST be removed. Note that if the content of the message includes another signature, such as a
580 signed SAML assertion, this embedded signature is not removed. However, the length of such a
581 message after encoding essentially precludes using this mechanism. Thus SAML protocol
582 messages that contain signed content SHOULD NOT be encoded using this mechanism.
- 583 2. The DEFLATE compression mechanism, as specified in [RFC1951] is then applied to the entire
584 remaining XML content of the original SAML protocol message.
- 585 3. The compressed data is subsequently base64-encoded according to the rules specified in IETF
586 RFC 2045 [RFC2045]. Linefeeds or other whitespace MUST be removed from the result.
- 587 4. The base-64 encoded data is then URL-encoded, and added to the URL as a query string
588 parameter which MUST be named `SAMLRequest` (if the message is a SAML request) or
589 `SAMLResponse` (if the message is a SAML response).
- 590 5. If RelayState data is to accompany the SAML protocol message, it MUST be URL-encoded and
591 placed in an additional query string parameter named `RelayState`.
- 592 6. If the original SAML protocol message was signed using an XML digital signature, a new signature
593 covering the encoded data as specified above MUST be attached using the rules stated below.

594 XML digital signatures are not directly URL-encoded according to the above rules, due to space concerns.
595 If the underlying SAML protocol message is signed with an XML signature [XMLSig], the URL-encoded
596 form of the message MUST be signed as follows:

- 597 1. The signature algorithm identifier MUST be included as an additional query string parameter,
598 named `SigAlg`. The value of this parameter MUST be a URI that identifies the algorithm used to
599 sign the URL-encoded SAML protocol message, specified according to [XMLSig] or whatever
600 specification governs the algorithm.
- 601 2. To construct the signature, a string consisting of the concatenation of the `RelayState` (if present),
602 `SigAlg`, and `SAMLRequest` (or `SAMLResponse`) query string parameters (each one URL-
603 encoded) is constructed in one of the following ways (ordered as below):
604

```
SAMLRequest=value&RelayState=value&SigAlg=value
```


605

```
SAMLResponse=value&RelayState=value&SigAlg=value
```
- 606 3. The resulting string of bytes is the octet string to be fed into the signature algorithm. Any other
607 content in the original query string is not included and not signed.
- 608 4. The signature value MUST be encoded using the base64 encoding (see RFC 2045 [RFC2045]) with
609 any whitespace removed, and included as a query string parameter named `Signature`. Note that
610 some characters in the base64-encoded signature value may themselves require URL-encoding
611 before being added.

612 5. The following signature algorithms (see [XMLSig]) and their URI representations MUST be
613 supported with this encoding mechanism:

- 614 • DSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#dsa-sha1>
- 615 • RSAwithSHA1 – <http://www.w3.org/2000/09/xmldsig#rsa-sha1>

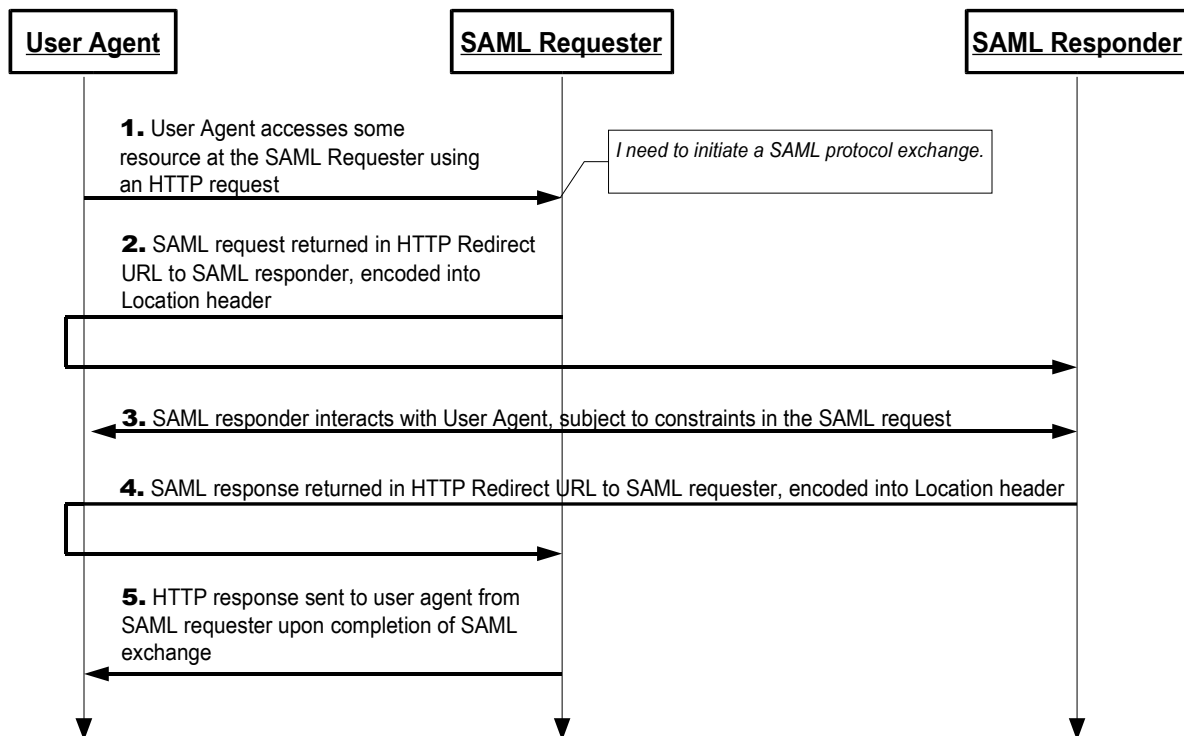
616 Note that when verifying signatures, the order of the query string parameters on the resulting URL to be
617 verified is not prescribed by this binding. The parameters may appear in any order. Before verifying a
618 signature, if any, the relying party MUST ensure that the parameter values to be verified are ordered as
619 required by the signing rules above.

620 Further, note that URL-encoding is not canonical; that is, there are multiple legal encodings for a given
621 value. The relying party MUST therefore perform the verification step using the original URL-encoded
622 values it received on the query string. It is not sufficient to re-encode the parameters after they have been
623 processed by software because the resulting encoding may not match the signer's encoding.

624 Finally, note that if there is no `RelayState` value, the entire parameter should be omitted from the
625 signature computation (and not included as an empty parameter name).

626 3.4.5 Message Exchange

627 The system model used for SAML conversations via this binding is a request-response model, but these
628 messages are sent to the user agent in an HTTP response and delivered to the message recipient in an
629 HTTP request. The HTTP interactions before, between, and after these exchanges take place is
630 unspecified. Both the SAML requester and the SAML responder are assumed to be HTTP responders.
631 See the following sequence diagram illustrating the messages exchanged.



632 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
633 processing the request, the system entity decides to initiate a SAML protocol exchange.

634 2. The system entity acting as a SAML requester responds to the HTTP request from the user agent in
635 step 1 by returning a SAML request. The SAML request is returned encoded into the HTTP

636 response's Location header, and the HTTP status MUST be either 303 or 302. The SAML requester
637 MAY include additional presentation and content in the HTTP response to facilitate the user agent's
638 transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user agent delivers the
639 SAML request by issuing an HTTP GET request to the SAML responder.

- 640 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
641 SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user
642 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
643 indicate the requester's level of willingness to permit this kind of interaction (for example, the
644 `IsPassive` attribute in `<samlp:AuthnRequest>`).
- 645 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
646 SAML requester. The SAML response is returned in the same fashion as described for the SAML
647 request in step 2.
- 648 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
649 user agent.

650 **3.4.5.1 HTTP and Caching Considerations**

651 HTTP proxies and the user agent intermediary should not cache SAML protocol messages. To ensure
652 this, the following rules SHOULD be followed.

653 When returning SAML protocol messages using HTTP 1.1, HTTP responders SHOULD:

- 654 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 655 • Include a `Pragma` header field set to "no-cache".

656 There are no other restrictions on the use of HTTP headers.

657 **3.4.5.2 Security Considerations**

658 The presence of the user agent intermediary means that the requester and responder cannot rely on the
659 transport layer for end-end authentication, integrity and confidentiality. URL-encoded messages MAY be
660 signed to provide origin authentication and integrity if the encoding method specifies a means for signing.

661 If the message is signed, the `Destination` XML attribute in the root SAML element of the protocol
662 message MUST contain the URL to which the sender has instructed the user agent to deliver the
663 message. The recipient MUST then verify that the value matches the location at which the message has
664 been received.

665 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
666 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
667 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS
668 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML
669 requester and responder.

670 Note also that URL-encoded messages may be exposed in a variety of HTTP logs as well as the HTTP
671 "Referer" header.

672 Before deployment, each combination of authentication, message integrity, and confidentiality
673 mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange, and
674 the deployment environment. See specific protocol processing rules in [SAMLCore], and the SAML
675 security considerations document [SAMLSecure] for a detailed discussion.

676 In general, this binding relies on message-level authentication and integrity protection via signing and
677 does not support confidentiality of messages from the user agent intermediary.

678 3.4.6 Error Reporting

679 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
680 return a SAML response message with a second-level <samlp:StatusCode> value of
681 urn:oasis:names:tc:SAML:2.0:status:RequestDenied.

682 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
683 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

684 For more information about SAML status codes, see the SAML assertions and protocols specification
685 [SAMLCore].

686 3.4.7 Metadata Considerations

687 Support for the HTTP Redirect binding SHOULD be reflected by indicating URL endpoints at which
688 requests and responses for a particular protocol or profile should be sent. Either a single endpoint or
689 distinct request and response endpoints MAY be supplied.

690 3.4.8 Example SAML Message Exchange Using HTTP Redirect

691 In this example, a <LogoutRequest> and <LogoutResponse> message pair is exchanged using the
692 HTTP Redirect binding.

693 First, here are the actual SAML protocol messages being exchanged:

```
694 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
695 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
696 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
697 21T19:00:49Z" Version="2.0">  
698 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
699 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
700 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
701 <samlp:SessionIndex>1</samlp:SessionIndex>  
702 </samlp:LogoutRequest>
```

```
703 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
704 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
705 ID="b0730d21b628110d8b7e004005b13a2b"  
706 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"  
707 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">  
708 <Issuer>https://ServiceProvider.com/SAML</Issuer>  
709 <samlp:Status>  
710 <samlp:StatusCode  
711 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
712 </samlp:Status>  
713 </samlp:LogoutResponse>
```

714 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
715 protocol exchange, the SAML requester returns the following HTTP response, containing a signed SAML
716 request message. The SAMLRequest parameter value is actually derived from the request message
717 above. The signature portion is only illustrative and not the result of an actual computation. Note that the
718 line feeds in the HTTP Location header below are an artifact of the document, and there are no line
719 feeds in the actual header value.

```
720 HTTP/1.1 302 Object Moved  
721 Date: 21 Jan 2004 07:00:49 GMT
```

```
722 Location:
723 https://ServiceProvider.com/SAML/SLO/Browser?SAMLRequest=fVFdS8MwFH0f7D%
724 2BUvGdNsq62oSsIQyhMESc%2B%2BJYlRbWpObeyvz3puv2IMjyFM7HPedyK1DdsZdb%2F%
725 2BEHfLFfgwVMt3RgTwzazIEJ72CFqRTnQWJWu7uH7dSLJjsg0ev%2FZFMlttiBWADtt6R%
726 2BSyJr9msiRH7070sCm31Mj%2Bo%2BC%
727 2B1KA5GLEWeZaogSQMw2MYBKodrIhjLKONU8FdeSsZkVr6T5M0GiHMjvWCknqZXZ2OoPx7kG
728 naGOuwXZ%2Fn4L9bY8NC%
729 2By4dulXpRXnxPcXizSZ58KfTeHujEWkNPZylsh9bAMYUjO2Uiy3jCpTCMo5M1StVjmN9SO1
730 50s191U6RV2Dp0vsLIy7NM7YU82r9B90PrvCf85W%2FwL8zSVQzAEAAA%3D%
731 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%
732 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
733 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
734 Content-Type: text/html; charset=iso-8859-1
```

735 After any unspecified interactions may have taken place, the SAML responder returns the HTTP response
736 below containing the signed SAML response message. Again, the `SAMLResponse` parameter value is
737 actually derived from the response message above. The signature portion is only illustrative and not the
738 result of an actual computation.

```
739 HTTP/1.1 302 Object Moved
740 Date: 21 Jan 2004 07:00:49 GMT
741 Location:
742 https://IdentityProvider.com/SAML/SLO/Response?SAMLResponse=fVFNa4QwEL0X%
743 2Bh8k912TaDUGFUp7EbZQ6rKH3mKcbQVNJBOX%2FvxaXQ9tYec0vHlv3nzKqIZ%2B1Af7YSf%
744 2FBjhagxB8Db1BuZQKMjkjrcIOpVEDoPRa1o8vB8n3VI7Oeqtt1bJbbJCB0c7a8j9XTBH9Vy
745 QhqYRbTlrEi4Yo61oUqA0pvShYZHiDQkqs411tAVpeZPqSagNOkrOas4zzcW55Zl14liJrTXi
746 BJVBr4wvCJ877ijbcXZkmaRUxtk7CU7gcB5mLu8pKVddvghd%
747 2Ben9iDIMA3CXtsOrs5euBbfxdgh%2F9snDK%2FEqW69Ye%2BUnvGL%2F8CfbQnBS%
748 2FQS3z4QLW9aT1oBIws0j%2FGoyAb9%2FV34Dw5k779IBAAA%
749 3D&RelayState=0043bfc1bc45110dae17004005b13a2b&SigAlg=http%3A%2F%
750 2Fwww.w3.org%2F200%2F09%2Fxmldsig%23rsa-
751 sha1&Signature=NOTAREALSIGNATUREBUTTHEREALONEWOULDGOHERE
752 Content-Type: text/html; charset=iso-8859-1
```

753 3.5 HTTP POST Binding

754 The HTTP POST binding defines a mechanism by which SAML protocol messages may be transmitted
755 within the base64-encoded content of an HTML form control.

756 This binding MAY be composed with the HTTP Redirect binding (see Section 3.4) and the HTTP Artifact
757 binding (see Section 3.6) to transmit request and response messages in a single protocol exchange using
758 two different bindings.

759 3.5.1 Required Information

760 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST

761 **Contact information:** security-services-comment@lists.oasis-open.org

762 **Description:** Given below.

763 **Updates:** Effectively replaces the binding aspects of the Browser/POST profile in SAML V1.1
764 [SAML11Bind].

765 3.5.2 Overview

766 The HTTP POST binding is intended for cases in which the SAML requester and responder need to
767 communicate using an HTTP user agent (as defined in HTTP 1.1 [RFC2616]) as an intermediary. This
768 may be necessary, for example, if the communicating parties do not share a direct path of communication.
769 It may also be needed if the responder requires an interaction with the user agent in order to fulfill the
770 request, such as when the user agent must authenticate to it.

771 Note that some HTTP user agents may have the capacity to play a more active role in the protocol
772 exchange and may support other bindings that use HTTP, such as the SOAP and Reverse SOAP
773 bindings. This binding assumes nothing apart from the capabilities of a common web browser.

774 **3.5.3 RelayState**

775 RelayState data MAY be included with a SAML protocol message transmitted with this binding. The value
776 MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the
777 message independent of any other protections that may or may not exist during message transmission.
778 Signing is not realistic given the space limitation, but because the value is exposed to third-party
779 tampering, the entity SHOULD ensure that the value has not been tampered with by using a checksum, a
780 pseudo-random value, or similar means.

781 If a SAML request message is accompanied by RelayState data, then the SAML responder MUST return
782 its SAML protocol response using a binding that also supports a RelayState mechanism, and it MUST
783 place the exact data it received with the request into the corresponding RelayState parameter in the
784 response.

785 If no such value is included with a SAML request message, or if the SAML response message is being
786 generated without a corresponding request, then the SAML responder MAY include RelayState data to be
787 interpreted by the recipient based on the use of a profile or prior agreement between the parties.

788 **3.5.4 Message Encoding**

789 Messages are encoded for use with this binding by encoding the XML into an HTML form control and are
790 transmitted using the HTTP POST method. A SAML protocol message is form-encoded by applying the
791 base-64 encoding rules to the XML representation of the message and placing the result in a hidden form
792 control within a form as defined by [HTML401] Section 17. The HTML document MUST adhere to the
793 XHTML specification, [XHTML]. The base64-encoded value MAY be line-wrapped at a reasonable length
794 in accordance with common practice.

795 If the message is a SAML request, then the form control MUST be named `SAMLRequest`. If the message
796 is a SAML response, then the form control MUST be named `SAMLResponse`. Any additional form controls
797 or presentation MAY be included but MUST NOT be required in order for the recipient to process the
798 message.

799 If a "RelayState" value is to accompany the SAML protocol message, it MUST be placed in an additional
800 hidden form control named `RelayState` within the same form with the SAML message.

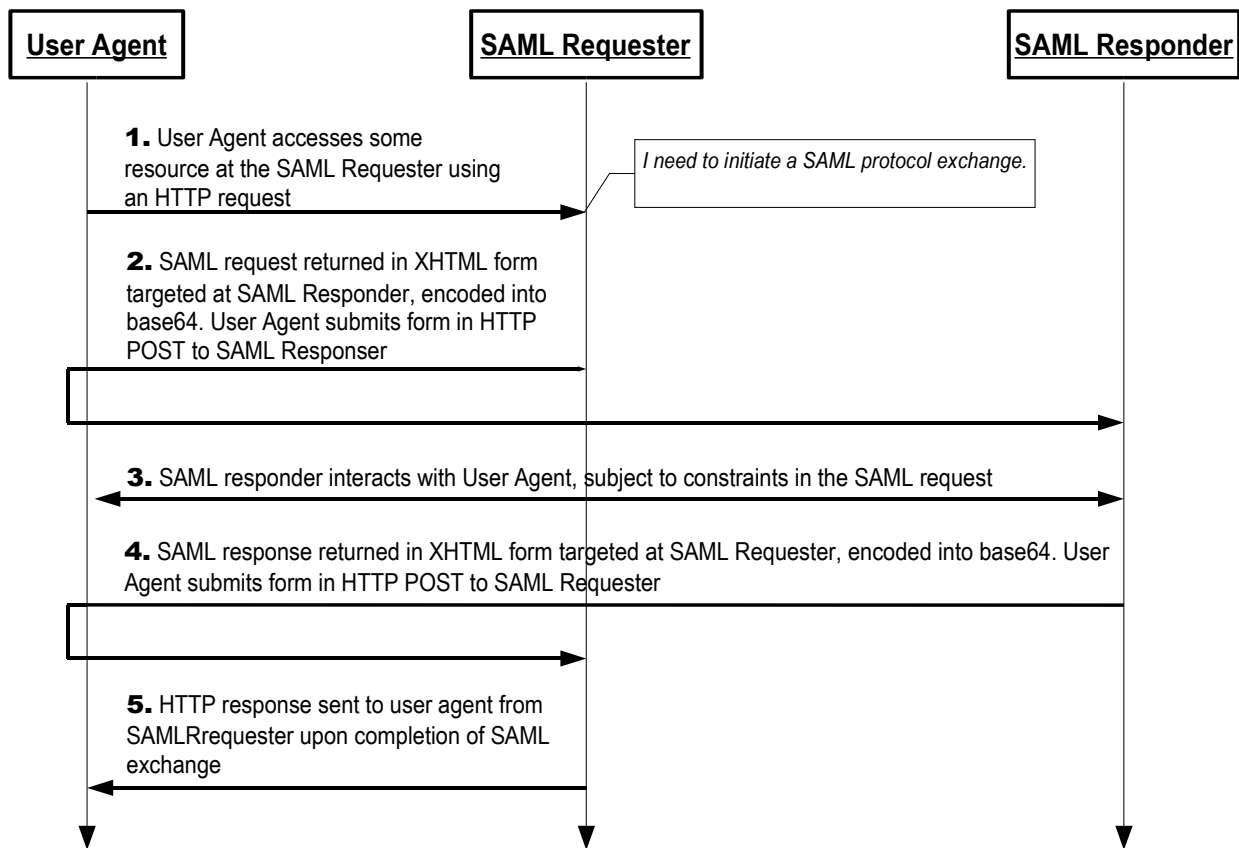
801 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
802 this binding to which the SAML message is to be delivered. The `method` attribute MUST be "POST".

803 Any technique supported by the user agent MAY be used to cause the submission of the form, and any
804 form content necessary to support this MAY be included, such as submit controls and client-side scripting
805 commands. However, the recipient MUST be able to process the message without regard for the
806 mechanism by which the form submission is initiated.

807 Note that any form control values included MUST be transformed so as to be safe to include in the
808 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

809 **3.5.5 Message Exchange**

810 The system model used for SAML conversations via this binding is a request-response model, but these
811 messages are sent to the user agent in an HTTP response and delivered to the message recipient in an
812 HTTP request. The HTTP interactions before, between, and after these exchanges take place is
813 unspecified. Both the SAML requester and responder are assumed to be HTTP responders. See the
814 following diagram illustrating the messages exchanged.



- 815 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
816 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 817 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
818 returning a SAML request. The request is returned in an XHTML document containing the form and
819 content defined in Section 3.5.4. The user agent delivers the SAML request by issuing an HTTP
820 POST request to the SAML responder.
- 821 3. In general, the SAML responder MAY respond to the SAML request by immediately returning a
822 SAML response or it MAY return arbitrary content to facilitate subsequent interaction with the user
823 agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to
824 indicate the requester's level of willingness to permit this kind of interaction (for example, the
825 `IsPassive` attribute in `<samlp:AuthnRequest>`).
- 826 4. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the
827 SAML requester. The SAML response is returned in the same fashion as described for the SAML
828 request in step 2.
- 829 5. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
830 user agent.

831 3.5.5.1 HTTP and Caching Considerations

832 HTTP proxies and the user agent intermediary should not cache SAML protocol messages. To ensure
833 this, the following rules SHOULD be followed.

834 When returning SAML protocol messages using HTTP 1.1, HTTP responders SHOULD:

- 835 • Include a `Cache-Control` header field set to "no-cache, no-store".

836 • Include a `Pragma` header field set to "no-cache".

837 There are no other restrictions on the use of HTTP headers.

838 3.5.5.2 Security Considerations

839 The presence of the user agent intermediary means that the requester and responder cannot rely on the
840 transport layer for end-end authentication, integrity or confidentiality protection and must authenticate the
841 messages received instead. SAML provides for a signature on protocol messages for authentication and
842 integrity for such cases. Form-encoded messages MAY be signed before the base64 encoding is applied.

843 If the message is signed, the `Destination` XML attribute in the root SAML element of the protocol
844 message MUST contain the URL to which the sender has instructed the user agent to deliver the
845 message. The recipient MUST then verify that the value matches the location at which the message has
846 been received.

847 This binding SHOULD NOT be used if the content of the request or response should not be exposed to
848 the user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
849 OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 [SSL3] or TLS
850 1.0 [RFC2246] SHOULD be used to protect the message in transit between the user agent and the SAML
851 requester and responder.

852 In general, this binding relies on message-level authentication and integrity protection via signing and
853 does not support confidentiality of messages from the user agent intermediary.

854 Note also that there is no mechanism defined to protect the integrity of the relationship between the SAML
855 protocol message and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair
856 of valid HTTP responses by switching the "RelayState" values associated with each SAML protocol
857 message. The individual "RelayState" and SAML message values can be integrity protected, but not the
858 combination. As a result, the producer and consumer of "RelayState" information MUST take care not to
859 associate sensitive state information with the "RelayState" value without taking additional precautions
860 (such as based on the information in the SAML message).

861 3.5.6 Error Reporting

862 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
863 return a response message with a second-level `<samlp:StatusCode>` value of
864 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

865 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
866 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

867 For more information about SAML status codes, see the SAML assertions and protocols specification
868 [SAMLCore].

869 3.5.7 Metadata Considerations

870 Support for the HTTP POST binding SHOULD be reflected by indicating URL endpoints at which requests
871 and responses for a particular protocol or profile should be sent. Either a single endpoint or distinct
872 request and response endpoints MAY be supplied.

873 3.5.8 Example SAML Message Exchange Using HTTP POST

874 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair is exchanged using the
875 HTTP POST binding.

876 First, here are the actual SAML protocol messages being exchanged:

991 **Contact information:** security-services-comment@lists.oasis-open.org

992 **Description:** Given below.

993 **Updates:** Effectively replaces the binding aspects of the Browser/Artifact profile in SAML V1.1
994 [SAML11Bind].

995 **3.6.2 Overview**

996 The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to
997 communicate using an HTTP user agent as an intermediary, but the intermediary's limitations preclude or
998 discourage the transmission of an entire message (or message exchange) through it. This may be for
999 technical reasons or because of a reluctance to expose the message content to the intermediary (and if
1000 the use of encryption is not practical).

1001 Note that because of the need to subsequently resolve the artifact using another synchronous binding,
1002 such as SOAP, a direct communication path must exist between the SAML message sender and recipient
1003 in the reverse direction of the artifact's transmission (the receiver of the message and artifact must be
1004 able to send a `<samlp:ArtifactResolve>` request back to the artifact issuer). The artifact issuer must
1005 also maintain state while the artifact is pending, which has implications for load-balanced environments.

1006 **3.6.3 Message Encoding**

1007 There are two methods of encoding an artifact for use with this binding. One is to encode the artifact into a
1008 URL parameter and the other is to place the artifact in an HTML form control. When URL encoding is
1009 used, the HTTP GET method is used to deliver the message, while POST is used with form encoding. All
1010 endpoints that support this binding MUST support both techniques.

1011 **3.6.3.1 RelayState**

1012 RelayState data MAY be included with a SAML artifact transmitted with this binding. The value MUST
1013 NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message
1014 independent of any other protections that may or may not exist during message transmission. Signing is
1015 not realistic given the space limitation, but because the value is exposed to third-party tampering, the
1016 entity SHOULD ensure that the value has not been tampered with by using a checksum, a pseudo-
1017 random value, or similar means.

1018 If an artifact that represents a SAML request is accompanied by RelayState data, then the SAML
1019 responder MUST return its SAML protocol response using a binding that also supports a RelayState
1020 mechanism, and it MUST place the exact data it received with the artifact into the corresponding
1021 RelayState parameter in the response.

1022 If no such value is included with an artifact representing a SAML request, or if the SAML response
1023 message is being generated without a corresponding request, then the SAML responder MAY include
1024 RelayState data to be interpreted by the recipient based on the use of a profile or prior agreement
1025 between the parties.

1026 **3.6.3.2 URL Encoding**

1027 To encode an artifact into a URL, the artifact value is URL-encoded and placed in a query string
1028 parameter named `SAMLart`.

1029 If a "RelayState" value is to accompany the SAML artifact, it MUST be URL-encoded and placed in an
1030 additional query string parameter named `RelayState`.

1031 3.6.3.3 Form Encoding

1032 A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by
1033 [HTML401], chapter 17. The HTML document MUST adhere to the XHTML specification, [XHTML]. The
1034 form control MUST be named `SAMLart`. Any additional form controls or presentation MAY be included but
1035 MUST NOT be required in order for the recipient to process the artifact.

1036 If a "RelayState" value is to accompany the SAML artifact, it MUST be placed in an additional hidden form
1037 control named `RelayState`, within the same form with the SAML message.

1038 The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using
1039 this binding to which the artifact is to be delivered. The `method` attribute MUST be set to "POST".

1040 Any technique supported by the user agent MAY be used to cause the submission of the form, and any
1041 form content necessary to support this MAY be included, such as submit controls and client-side scripting
1042 commands. However, the recipient MUST be able to process the artifact without regard for the
1043 mechanism by which the form submission is initiated.

1044 Note that any form control values included MUST be transformed so as to be safe to include in the
1045 XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

1046 3.6.4 Artifact Format

1047 With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used
1048 without affecting the binding. The important characteristics are the ability of an artifact receiver to identify
1049 the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

1050 The general format of any artifact includes a mandatory two-byte artifact type code and a two-byte index
1051 value identifying a specific endpoint of the artifact resolution service of the issuer, as follows:

```
1052 SAML_artifact      := B64( TypeCode EndpointIndex RemainingArtifact )
1053 TypeCode           := Byte1Byte2
1054 EndpointIndex      := Byte1Byte2
```

1055 The notation `B64(TypeCode EndpointIndex RemainingArtifact)` stands for the application of
1056 the base64 [RFC2045] transformation to the catenation of the `TypeCode`, `EndpointIndex`, and
1057 `RemainingArtifact`.

1058 The following practices are RECOMMENDED for the creation of SAML artifacts:

- 1059 • Each issuer is assigned an identifying URI, also known as the issuer's entity (or provider) ID. See
1060 Section 8.3.6 of [SAMLCore] for a discussion of this kind of identifier.
- 1061 • The issuer constructs the `SourceID` component of the artifact by taking the SHA-1 hash of the
1062 identification URL. The hash value is NOT encoded into hexadecimal.
- 1063 • The `MessageHandle` value is constructed from a cryptographically strong random or
1064 pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of
1065 values of at least 16 bytes in size. These values should be padded as needed to a total length of 20
1066 bytes.

1067 The following describes the single artifact type defined by SAML V2.0.

1068 3.6.4.1 Required Information

1069 **Identification:** urn:oasis:names:tc:SAML:2.0:artifact-04

1070 **Contact information:** security-services-comment@lists.oasis-open.org

1071 **Description:** Given below.

1072 **Updates:** None.

1073 3.6.4.2 Format Details

1074 SAML V2.0 defines an artifact type of type code 0x0004. This artifact type is defined as follows:

1075	TypeCode	:= 0x0004
1076	RemainingArtifact	:= SourceID MessageHandle
1077	SourceID	:= 20-byte_sequence
1078	MessageHandle	:= 20-byte_sequence

1079 SourceID is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and the
1080 set of possible resolution endpoints.

1081 It is assumed that the destination site will maintain a table of SourceID values as well as one or more
1082 indexed URL endpoints (or addresses) for the corresponding SAML responder. The SAML metadata
1083 specification [SAMLMeta] MAY be used for this purpose. On receiving the SAML artifact, the receiver
1084 determines if the SourceID belongs to a known artifact issuer and obtains the location of the SAML
1085 responder using the EndpointIndex before sending a SAML <samlp:ArtifactResolve> message
1086 to it.

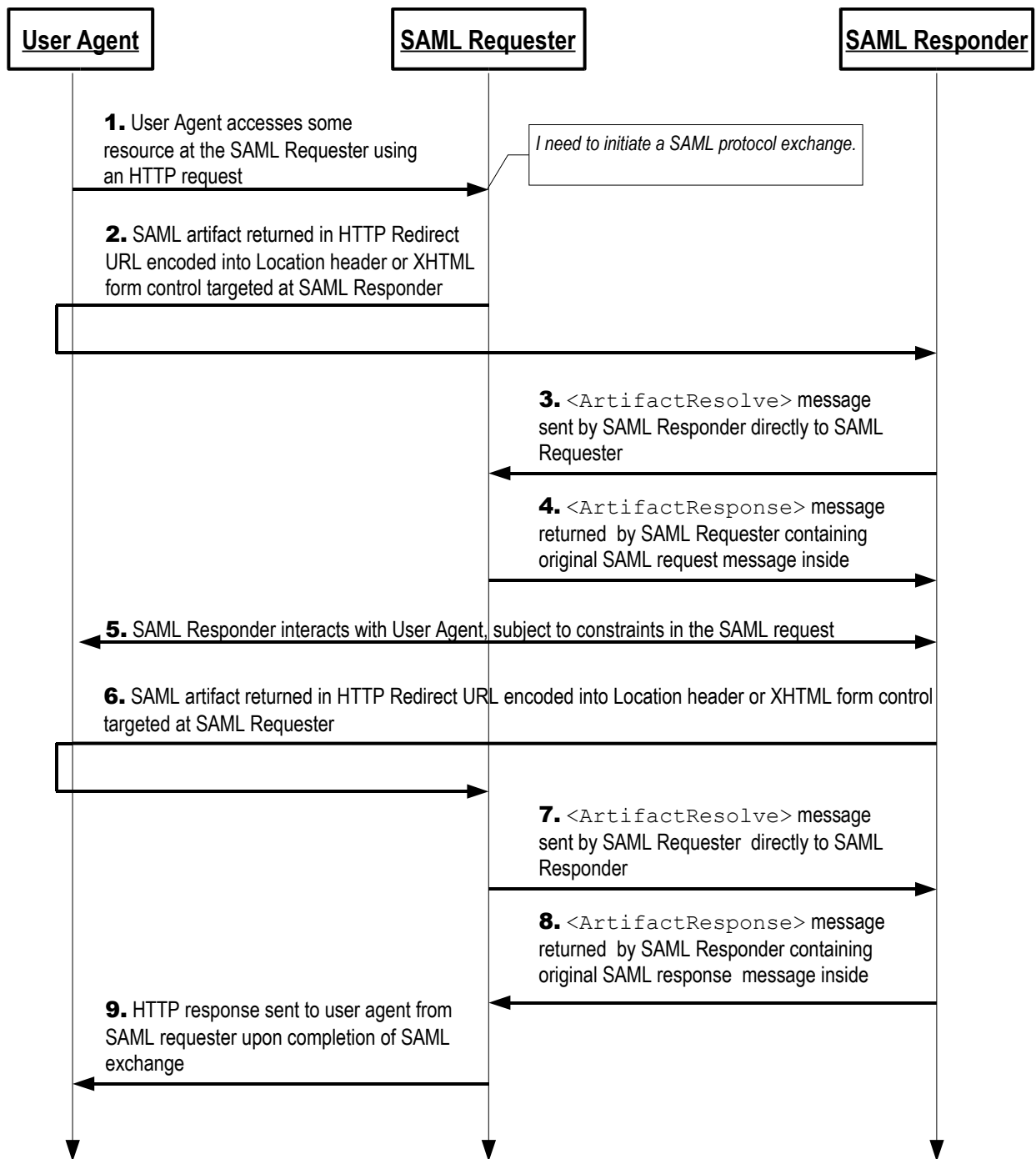
1087 Any two artifact issuers with a common receiver MUST use distinct SourceID values. Construction of
1088 MessageHandle values is governed by the principle that they SHOULD have no predictable relationship
1089 to the contents of the referenced message at the issuing site and it MUST be infeasible to construct or
1090 guess the value of a valid, outstanding message handle.

1091 3.6.5 Message Exchange

1092 The system model used for SAML conversations by means of this binding is a request-response model in
1093 which an artifact reference takes the place of the actual message content, and the artifact reference is
1094 sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request.
1095 The HTTP interactions before, between, and after these exchanges take place is unspecified. Both the
1096 SAML requester and responder are assumed to be HTTP responders.

1097 Additionally, it is assumed that on receipt of an artifact by way of the user agent, the recipient invokes a
1098 separate, direct exchange with the artifact issuer using the Artifact Resolution Protocol defined in
1099 [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an
1100 intermediary, such as the SOAP binding. On the successful acquisition of a SAML protocol message, the
1101 artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the
1102 message is a response).

1103 Issuing and delivering an artifact, along with the subsequent resolution step, constitutes half of the overall
1104 SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol
1105 exchange. A binding composable with it, such as the HTTP Redirect (see Section 3.4) or POST (see
1106 Section 3.5) binding, MAY be used to carry the other half of the exchange. The following sequence
1107 assumes that the artifact binding is used for both halves. See the diagram below illustrating the messages
1108 exchanged.



- 1109 1. Initially, the user agent makes an arbitrary HTTP request to a system entity. In the course of
 1110 processing the request, the system entity decides to initiate a SAML protocol exchange.
- 1111 2. The system entity acting as a SAML requester responds to an HTTP request from the user agent by
 1112 returning an artifact representing a SAML request.
- 1113 • If URL-encoded, the artifact is returned encoded into the HTTP response's Location
 1114 header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY
 1115 include additional presentation and content in the HTTP response to facilitate the user
 1116 agent's transmission of the message, as defined in HTTP 1.1 [RFC2616]. The user

1117 agent delivers the artifact by issuing an HTTP GET request to the SAML responder.

1118 • If form-encoded, then the artifact is returned in an XHTML document containing the
1119 form and content defined in Section 3.6.3.3. The user agent delivers the artifact by
1120 issuing an HTTP POST request to the SAML responder.

1121 3. The SAML responder determines the SAML requester by examining the artifact (the exact process
1122 depends on the type of artifact), and issues a `<samlp:ArtifactResolve>` request containing
1123 the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles.

1124 4. Assuming the necessary conditions are met, the SAML requester returns a
1125 `<samlp:ArtifactResponse>` containing the original SAML request message it wishes the
1126 SAML responder to process.

1127 5. In general, the SAML responder MAY respond to the SAML request by immediately returning a
1128 SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent
1129 necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate
1130 the requester's level of willingness to permit this kind of interaction (for example, the `IsPassive`
1131 attribute in `<samlp:AuthnRequest>`).

1132 6. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the
1133 SAML requester. The SAML response artifact is returned in the same fashion as described for the
1134 SAML request artifact in step 2. The SAML requester determines the SAML responder by examining
1135 the artifact, and issues a `<samlp:ArtifactResolve>` request containing the artifact to the SAML
1136 responder using a direct SAML binding, as in step 3.

1137 7. Assuming the necessary conditions are met, the SAML responder returns a
1138 `<samlp:ArtifactResponse>` containing the SAML response message it wishes the requester to
1139 process, as in step 4.

1140 8. Upon receiving the SAML response, the SAML requester returns an arbitrary HTTP response to the
1141 user agent.

1142 **3.6.5.1 HTTP and Caching Considerations**

1143 HTTP proxies and the user agent intermediary should not cache SAML artifacts. To ensure this, the
1144 following rules SHOULD be followed.

1145 When returning SAML artifacts using HTTP 1.1, HTTP responders SHOULD:

- 1146 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1147 • Include a `Pragma` header field set to "no-cache".

1148 There are no other restrictions on the use of HTTP headers.

1149 **3.6.5.2 Security Considerations**

1150 This binding uses a combination of indirect transmission of a message reference followed by a direct
1151 exchange to return the actual message. As a result, the message reference (artifact) need not itself be
1152 authenticated or integrity protected, but the callback request/response exchange that returns the actual
1153 message MAY be mutually authenticated and integrity protected, depending on the environment of use.

1154 If the actual SAML protocol message is intended for a specific recipient, then the artifact's issuer MUST
1155 authenticate the sender of the subsequent `<samlp:ArtifactResolve>` message before returning the
1156 actual message.

1157 The transmission of an artifact to and from the user agent SHOULD be protected with confidentiality; SSL
1158 3.0 [SSL3] or TLS 1.0 [RFC2246] SHOULD be used. The callback request/response exchange that
1159 returns the actual message MAY be protected, depending on the environment of use.

1160 In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other
1161 security measures to the callback request/response that returns the actual message. All artifacts MUST
1162 have a single-use semantic enforced by the artifact issuer.

1163 Furthermore, it is RECOMMENDED that artifact receivers also enforce a single-use semantic on the
1164 artifact values they receive, to prevent an attacker from interfering with the resolution of an artifact by a
1165 user agent and then resubmitting it to the artifact receiver. If an attempt to resolve an artifact does not
1166 complete successfully, the artifact SHOULD be placed into a blocked artifact list for a period of time that
1167 exceeds a reasonable acceptance period during which the artifact issuer would resolve the artifact.

1168 Note also that there is no mechanism defined to protect the integrity of the relationship between the
1169 artifact and the "RelayState" value, if any. That is, an attacker can potentially recombine a pair of valid
1170 HTTP responses by switching the "RelayState" values associated with each artifact. As a result, the
1171 producer/consumer of "RelayState" information MUST take care not to associate sensitive state
1172 information with the "RelayState" value without taking additional precautions (such as based on the
1173 information in the SAML protocol message retrieved via artifact).

1174 **3.6.6 Error Reporting**

1175 A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
1176 return a response message with a second-level `<samlp:StatusCode>` value of
1177 `urn:oasis:names:tc:SAML:2.0:status:RequestDenied`.

1178 HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
1179 failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

1180 If the issuer of an artifact receives a `<samlp:ArtifactResolve>` message that it can understand, it
1181 MUST return a `<samlp:ArtifactResponse>` with a `<samlp:StatusCode>` value of
1182 `urn:oasis:names:tc:SAML:2.0:status:Success`, even if it does not return the corresponding
1183 message (for example because the artifact requester is not authorized to receive the message or the
1184 artifact is no longer valid).

1185 For more information about SAML status codes, see the SAML assertions and protocols specification
1186 [SAMLCore].

1187 **3.6.7 Metadata Considerations**

1188 Support for the HTTP Artifact binding SHOULD be reflected by indicating URL endpoints at which
1189 requests and responses for a particular protocol or profile should be sent. Either a single endpoint or
1190 distinct request and response endpoints MAY be supplied. One or more indexed endpoints for processing
1191 `<samlp:ArtifactResolve>` messages SHOULD also be described.

1192 **3.6.8 Example SAML Message Exchange Using HTTP Artifact**

1193 In this example, a `<LogoutRequest>` and `<LogoutResponse>` message pair is exchanged using the
1194 HTTP Artifact binding, with the artifact resolution taking place using the SOAP binding bound to HTTP.

1195 First, here are the actual SAML protocol messages being exchanged:

```
1196 <samlp:LogoutRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"  
1197 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"  
1198 ID="d2b7c388cec36fa7c39c28fd298644a8" IssueInstant="2004-01-  
1199 21T19:00:49Z" Version="2.0">  
1200 <Issuer>https://IdentityProvider.com/SAML</Issuer>  
1201 <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-  
1202 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>  
1203 <samlp:SessionIndex>1</samlp:SessionIndex>  
1204 </samlp:LogoutRequest>
```



```

1205 <samlp:LogoutResponse xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1206 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1207 ID="b0730d21b628110d8b7e004005b13a2b"
1208 InResponseTo="d2b7c388cec36fa7c39c28fd298644a8"
1209 IssueInstant="2004-01-21T19:00:49Z" Version="2.0">
1210 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1211 <samlp:Status>
1212 <samlp:StatusCode
1213 Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1214 </samlp:Status>
1215 </samlp:LogoutResponse>

```

1216 The initial HTTP request from the user agent in step 1 is not defined by this binding. To initiate the logout
1217 protocol exchange, the SAML requester returns the following HTTP response, containing a SAML artifact.
1218 Note that the line feeds in the HTTP Location header below are a result of document formatting, and
1219 there are no line feeds in the actual header value.

```

1220 HTTP/1.1 302 Object Moved
1221 Date: 21 Jan 2004 07:00:49 GMT
1222 Location:
1223 https://ServiceProvider.com/SAML/SLO/Browser?SAMLart=AAQAADWNEw5VT47wcO4z
1224 X%2FiEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU%
1225 3D&RelayState=0043bfclbc45110dae17004005b13a2b
1226 Content-Type: text/html; charset=iso-8859-1

```

1227 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1228 Resolution protocol and the SOAP binding in steps 3 and 4, as follows:

1229 Step 3:

```

1230 POST /SAML/Artifact/Resolve HTTP/1.1
1231 Host: IdentityProvider.com
1232 Content-Type: text/xml
1233 Content-Length: nnn
1234 SOAPAction: http://www.oasis-open.org/committees/security
1235 <SOAP-ENV:Envelope
1236 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1237 <SOAP-ENV:Body>
1238 <samlp:ArtifactResolve
1239 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1240 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1241 ID="_6c3a4f8b9c2d" Version="2.0"
1242 IssueInstant="2004-01-21T19:00:49Z">
1243 <Issuer>https://ServiceProvider.com/SAML</Issuer>
1244 <Artifact>
1245 AAQAADWNEw5VT47wcO4zX/iEzMmFQvGknDfws2ZtqSGdkNSbsW1cmVR0bzU=
1246 </Artifact>
1247 </samlp:ArtifactResolve>
1248 </SOAP-ENV:Body>
1249 </SOAP-ENV:Envelope>

```

1250 Step 4:

```

1251 HTTP/1.1 200 OK
1252 Date: 21 Jan 2004 07:00:49 GMT
1253 Content-Type: text/xml
1254 Content-Length: nnnn
1255 <SOAP-ENV:Envelope
1256 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1257 <SOAP-ENV:Body>
1258 <samlp:ArtifactResponse
1259 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1260 xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1261 ID="FQvGknDfws2Z" Version="2.0"
1262 InResponseTo="_6c3a4f8b9c2d"
1263 IssueInstant="2004-01-21T19:00:49Z">
1264 <Issuer>https://IdentityProvider.com/SAML</Issuer>
1265 <samlp:Status>

```

```

1266         <samlp:StatusCode
1267         Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1268     </samlp:Status>
1269     <samlp:LogoutRequest ID="d2b7c388cec36fa7c39c28fd298644a8"
1270     IssueInstant="2004-01-21T19:00:49Z"
1271     Version="2.0">
1272         <Issuer>https://IdentityProvider.com/SAML</Issuer>
1273         <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
1274 format:persistent">005a06e0-ad82-110d-a556-004005b13a2b</NameID>
1275         <samlp:SessionIndex>1</samlp:SessionIndex>
1276     </samlp:LogoutRequest>
1277 </samlp:ArtifactResponse>
1278 </SOAP-ENV:Body>
1279 </SOAP-ENV:Envelope>

```

1280 After any unspecified interactions may have taken place, the SAML responder returns a second SAML
1281 artifact in its HTTP response in step 6:

```

1282 HTTP/1.1 302 Object Moved
1283 Date: 21 Jan 2004 07:05:49 GMT
1284 Location:
1285 https://IdentityProvider.com/SAML/SLO/Response?SAMLart=AAQAAFZGIZXv5%
1286 2BQaBaE5qYurHWJO1nAgLASqfnyidHIggbFU0mlSGFTyQiPc%
1287 3D&RelayState=0043bfc1bc45110dae17004005b13a2b
1288 Content-Type: text/html; charset=iso-8859-1

```

1289 The SAML responder then resolves the artifact it received into the actual SAML request using the Artifact
1290 Resolution protocol and the SOAP binding in steps 7 and 8, as follows:

1291 Step 7:

```

1292 POST /SAML/Artifact/Resolve HTTP/1.1
1293 Host: ServiceProvider.com
1294 Content-Type: text/xml
1295 Content-Length: nnn
1296 SOAPAction: http://www.oasis-open.org/committees/security
1297 <SOAP-ENV:Envelope
1298   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1299   <SOAP-ENV:Body>
1300     <samlp:ArtifactResolve
1301     xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1302     xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1303     ID="_ec36fa7c39" Version="2.0"
1304     IssueInstant="2004-01-21T19:05:49Z">
1305       <Issuer>https://IdentityProvider.com/SAML</Issuer>
1306       <Artifact>
1307         AAQAAFZGIZXv5+QaBaE5qYurHWJO1nAgLASqfnyidHIggbFU0mlSGFTyQiPc=
1308       </Artifact>
1309     </samlp:ArtifactResolve>
1310   </SOAP-ENV:Body>
1311 </SOAP-ENV:Envelope>

```

1312 Step 8:

```

1313 HTTP/1.1 200 OK
1314 Date: 21 Jan 2004 07:05:49 GMT
1315 Content-Type: text/xml
1316 Content-Length: nnnn
1317 <SOAP-ENV:Envelope
1318   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
1319   <SOAP-ENV:Body>
1320     <samlp:ArtifactResponse
1321     xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
1322     xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
1323     ID="_FQvGknDfws2Z" Version="2.0"
1324     InResponseTo="_ec36fa7c39"
1325     IssueInstant="2004-01-21T19:05:49Z">
1326     <Issuer>https://ServiceProvider.com/SAML</Issuer>

```

```
1327     <samlp:Status>
1328         <samlp:StatusCode
1329         Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1330     </samlp:Status>
1331     <samlp:LogoutResponse ID="_b0730d21b628110d8b7e004005b13a2b"
1332     InResponseTo="_d2b7c388cec36fa7c39c28fd298644a8"
1333     IssueInstant="2004-01-21T19:05:49Z"
1334     Version="2.0">
1335         <Issuer>https://ServiceProvider.com/SAML</Issuer>
1336         <samlp:Status>
1337             <samlp:StatusCode
1338             Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
1339         </samlp:Status>
1340     </samlp:LogoutResponse>
1341 </samlp:ArtifactResponse>
1342 </SOAP-ENV:Body>
1343 </SOAP-ENV:Envelope>
```

1344 3.7 SAML URI Binding

1345 URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML
1346 request/response binding, but rather supports the encapsulation of a `<samlp:AssertionIDRequest>`
1347 message with a single `<saml:AssertionIDRef>` into the resolution of a URI. The result of a successful
1348 request is a SAML `<saml:Assertion>` element (but not a complete SAML response).

1349 Like SOAP, URI resolution can occur over multiple underlying transports. This binding has transport-
1350 independent aspects, but also calls out the use of HTTP with SSL 3.0 [SSL3] or TLS 1.0 [RFC2246] as
1351 REQUIRED (mandatory to implement).

1352 3.7.1 Required Information

1353 **Identification:** urn:oasis:names:tc:SAML:2.0:bindings:URI

1354 **Contact information:** security-services-comment@lists.oasis-open.org

1355 **Description:** Given below.

1356 **Updates:** None

1357 3.7.2 Protocol-Independent Aspects of the SAML URI Binding

1358 The following sections define aspects of the SAML URI binding that are independent of the underlying
1359 transport protocol of the URI resolution process.

1360 3.7.2.1 Basic Operation

1361 A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a
1362 message containing the assertion, or a transport-specific error. The specific format of the message
1363 depends on the underlying transport protocol. If the transport protocol permits the returned content to be
1364 described, such as HTTP 1.1 [RFC2616], then the assertion MAY be encoded in whatever format is
1365 permitted. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or
1366 transformed into an XML serialization of the assertion.

1367 It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML
1368 assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently
1369 reference the same assertion, if any.

1370 **3.7.3 Security Considerations**

1371 Indirect use of a SAML assertion presents dangers if the binding of the reference to the result is not
1372 secure. The particular threats and their severity depend on the use to which the assertion is being put. In
1373 general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the
1374 requester can be certain of the identity of the responder and that the contents have not been modified in
1375 transit.

1376 It is often not sufficient that the assertion itself be signed, because URI references are by their nature
1377 somewhat opaque to the requester. The requester SHOULD have independent means to ensure that the
1378 assertion returned is actually the one that is represented by the URI; this is accomplished by both
1379 authenticating the responder and relying on the integrity of the response.

1380 **3.7.4 MIME Encapsulation**

1381 For resolution protocols that support MIME as a content description and packaging mechanism, the
1382 resulting assertion SHOULD be returned as a MIME entity of type `application/samlassertion+xml`,
1383 as defined by [SAMLmime].

1384 **3.7.5 Use of HTTP URIs**

1385 A SAML authority that claims conformance to the SAML URI binding MUST implement support for HTTP.
1386 This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, and
1387 error reporting.

1388 **3.7.5.1 URI Syntax**

1389 In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the
1390 SAML authority responsible for the reference creates the message containing it. However, authorities
1391 MUST support a URL endpoint at which an HTTP request can be sent with a single query string
1392 parameter named `ID`. There MUST be no query string in the endpoint URL itself independent of this
1393 parameter.

1394 For example, if the documented endpoint at an authority is "https://saml.example.edu/assertions", a
1395 request for an assertion with an `ID` of `abcde` can be sent to:

1396 `https://saml.example.edu/assertions?ID=abcde`

1397 Note that the use of wildcards is not allowed for such ID queries.

1398 **3.7.5.2 HTTP and Caching Considerations**

1399 HTTP proxies MUST NOT cache SAML assertions. To ensure this, the following rules SHOULD be
1400 followed.

1401 When returning SAML assertions using HTTP 1.1, HTTP responders SHOULD:

- 1402 • Include a `Cache-Control` header field set to "no-cache, no-store".
- 1403 • Include a `Pragma` header field set to "no-cache".

1404 **3.7.5.3 Security Considerations**

1405 RFC 2617 [RFC2617] describes possible attacks in the HTTP environment when basic or message-digest
1406 authentication schemes are used.

1407 Use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246] is STRONGLY RECOMMENDED as a means of
1408 authentication, integrity protection, and confidentiality.

1409 **3.7.5.4 Error Reporting**

1410 As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result
1411 of a request. For example, a SAML responder that refuses to perform a message exchange with the
1412 SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to
1413 the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of
1414 the HTTP body is not significant.

1415 **3.7.5.5 Metadata Considerations**

1416 Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which
1417 requests for arbitrary assertions are to be sent.

1418 **3.7.5.6 Example SAML Message Exchange Using an HTTP URI**

1419 Following is an example of a request for an assertion.

```
1420 GET /SamlService?ID=abcde HTTP/1.1  
1421 Host: www.example.com
```

1422 Following is an example of the corresponding response, which supplies the requested assertion.

```
1423 HTTP/1.1 200 OK  
1424 Content-Type: application/samlassertion+xml  
1425 Cache-Control: no-cache, no-store  
1426 Pragma: no-cache  
1427 Content-Length: nnnn  
  
1428 <saml:Assertion ID="abcde" ...>  
1429 ...  
1430 </saml:Assertion>
```

4 References

1431

- 1432 **[HTML401]** D. Raggett et al. *HTML 4.01 Specification*. World Wide Web Consortium
1433 Recommendation, December 1999. See <http://www.w3.org/TR/html4>.
- 1434 **[Liberty]** The Liberty Alliance Project. See <http://www.projectliberty.org>.
- 1435 **[PAOS]** R. Aarts. *Liberty Reverse HTTP Binding for SOAP Specification Version 1.0*.
1436 Liberty Alliance Project, 2003. See [https://www.projectliberty.org/specs/liberty-](https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf)
1437 [paos-v1.0.pdf](https://www.projectliberty.org/specs/liberty-paos-v1.0.pdf).
- 1438 **[RFC1750]** D. Eastlake et al. *Randomness Recommendations for Security*. IETF RFC 1750,
1439 December 1994. See <http://www.ietf.org/rfc/rfc1750.txt>.
- 1440 **[RFC2045]** N. Freed et al. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of*
1441 *Internet Message Bodies*, IETF RFC 2045, November 1996. See
1442 <http://www.ietf.org/rfc/rfc2045.txt>.
- 1443 **[RFC2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF
1444 RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>.
- 1445 **[RFC2246]** T. Dierks et al. *The TLS Protocol Version 1.0*. IETF RFC 2246, January 1999.
1446 See <http://www.ietf.org/rfc/rfc2246.txt>.
- 1447 **[RFC2279]** F. Yergeau. *UTF-8, a transformation format of ISO 10646*. IETF RFC 2279,
1448 January 1998. See <http://www.ietf.org/rfc/rfc2279.txt>.
- 1449 **[RFC2616]** R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June
1450 1999. See <http://www.ietf.org/rfc/rfc2616.txt>.
- 1451 **[RFC2617]** J. Franks et al. *HTTP Authentication: Basic and Digest Access Authentication*.
1452 IETF RFC 2617, June 1999. See <http://www.ietf.org/rfc/rfc2617.txt>.
- 1453 **[SAML11Bind]** E. Maler et al. *Bindings and Profiles for the OASIS Security Assertion Markup*
1454 *Language (SAML)*. OASIS, September 2003. Document ID oasis-sstc-saml-
1455 bindings-1.1. See <http://www.oasis-open.org/committees/security/>.
- 1456 **[SAMLConform]** P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion*
1457 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
1458 conformance-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 1459 **[SAMLCore]** S. Cantor et al. *Assertions and Protocols for the OASIS Security Assertion*
1460 *Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-
1461 core-2.0-os. See <http://www.oasis-open.org/committees/security/>.
- 1462 **[SAMLGloss]** J. Hodges et al. *Glossary for the OASIS Security Assertion Markup Language*
1463 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-glossary-2.0-os.
1464 See <http://www.oasis-open.org/committees/security/>.
- 1465 **[SAMLMeta]** S. Cantor et al. *Metadata for the OASIS Security Assertion Markup Language*
1466 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-metadata-2.0-os.
1467 See <http://www.oasis-open.org/committees/security/>.
- 1468 **[SAMLmime]** application/saml+xml Media Type Registration, IETF Internet-Draft,
1469 <http://www.ietf.org/internet-drafts/draft-hodges-saml-mediatype-01.txt>.
- 1470 **[SAMLProfile]** S. Cantor et al. *Profiles for the OASIS Security Assertion Markup Language*
1471 *(SAML) V2.0*. OASIS SSTC, March 2005. Document ID saml-profiles-2.0-os. See
1472 <http://www.oasis-open.org/committees/security/>.
- 1473 **[SAMLSecure]** F. Hirsch et al. *Security and Privacy Considerations for the OASIS Security*
1474 *Assertion Markup Language (SAML) V2.0*. OASIS SSTC, March 2005. Document
1475 ID saml-sec-consider-2.0-os. See [http://www.oasis-](http://www.oasis-open.org/committees/security/)
1476 [open.org/committees/security/](http://www.oasis-open.org/committees/security/).
- 1477 **[SOAP11]** D. Box et al. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web

1478 Consortium Note, May 2000. See <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
1479

1480 **[SOAP-PRIMER]** N. Mitra. *SOAP Version 1.2 Part 0: Primer*. World Wide Web Consortium
1481 Recommendation, June 2003. See <http://www.w3.org/TR/soap12-part0/>,

1482 **[SSL3]** A. Frier et al. *The SSL 3.0 Protocol*. Netscape Communications Corp, November
1483 1996.

1484 **[SSTCWeb]** OASIS Security Services Technical Committee website, [http://www.oasis-
1485 open.org/committees/security](http://www.oasis-open.org/committees/security).

1486 **[XHTML]** *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. World
1487 Wide Web Consortium Recommendation, August 2002. See
1488 <http://www.w3.org/TR/xhtml1/>.

1489 **[XMLSig]** D. Eastlake et al. *XML-Signature Syntax and Processing*. World Wide Web
1490 Consortium Recommendation, February 2002. See
1491 <http://www.w3.org/TR/xmlsig-core/>.

1492 **Appendix A. Registration of MIME media type**
1493 **application/samlassertion+xml**

1494 **Introduction**

1495 This document defines a MIME media type -- `application/samlassertion+xml` -- for use
1496 with the XML serialization of SAML (Security Assertion Markup Language) assertions.

1497 The SAML specification sets -- [SAMLv1.0], [SAMLv1.1], [SAMLv2.0] -- are work products of the
1498 OASIS Security Services Technical Committee [SSTC]. The SAML specifications define XML-
1499 based constructs with which one may make, and convey, security assertions. Using SAML, one
1500 can assert that an authentication event pertaining to some subject has occurred and convey said
1501 assertion to a relying party, for example.

1502 SAML assertions, which are explicitly versioned, are defined by [SAMLv1Core], [SAMLv11Core],
1503 and [SAMLv2Core].

1504 **MIME media type name**

1505 `application`

1506 **MIME subtype name**

1507 `samlassertion+xml`

1508 **Required parameters**

1509 None

1510 **Optional parameters**

1511 `charset`

1512 Same as `charset` parameter of `application/xml` [RFC3023].

1513 **Encoding considerations**

1514 Same as for `application/xml` [RFC3023].

1515 **Security considerations**

1516 Per their specification, `samlassertion+xml`-typed objects do not contain executable content.
1517 However, SAML assertions are XML-based objects [XML]. As such, they have all of the general
1518 security considerations presented in Section 10 of [RFC3023], as well as additional ones, since
1519 they are explicit security objects. For example, `samlassertion+xml`-typed objects will often
1520 contain data that may identify or pertain to a natural person, and may be used as a basis for
1521 sessions and access control decisions.

1522 To counter potential issues, `samlassertion+xml`-typed objects contain data that should be
1523 signed appropriately by the sender. Any such signature must be verified by the recipient of the
1524 data - both as a valid signature, and as being the signature of the sender. Issuers of
1525 `samlassertion+xml`-typed objects containing SAMLv2 assertions may also encrypt all, or
1526 portions of, the assertions (see [SAMLv2Core]).

1527 In addition, SAML profiles and protocol bindings specify use of secure channels as appropriate.

1528 [SAMLv2.0] incorporates various privacy-protection techniques in its design. For example: opaque
1529 handles, specific to interactions between specific system entities, may be assigned to subjects.
1530 The handles are mappable to wider-context identifiers (e.g. email addresses, account identifiers,
1531 etc) by only the specific parties.

1532 For a more detailed discussion of SAML security considerations and specific security-related
1533 design techniques, please refer to the SAML specifications listed in the below bibliography. The
1534 specifications containing security-specific information have been explicitly listed for each version
1535 of SAML.

1536 Interoperability considerations

1537 SAML assertions are explicitly versioned. Relying parties should ensure that they observe
1538 assertion version information and behave accordingly. See chapters on SAML Versioning in
1539 [SAMLv1Core], [SAMLv11Core], or [SAMLv2Core], as appropriate.

1540 Published specification

1541 [SAMLv2Bind] explicitly specifies use of the `application/samlassertion+xml` MIME media
1542 type. However, it is conceivable that non-SAMLv2 assertions (i.e., SAMLv1 and/or SAMLv1.1)
1543 might in practice be conveyed using SAMLv2 bindings.

1544 Applications which use this media type

1545 Potentially any application implementing SAML, as well as those applications implementing
1546 specifications based on SAML, e.g. those available from the Liberty Alliance [LAP].

1547 Additional information

1548 Magic number(s)

1549 In general, the same as for `application/xml` [RFC3023]. In particular, the XML root element of the
1550 returned object will have a namespace-qualified name with:

- 1551 – a local name of: `Assertion`
- 1552 – a namespace URI of: one of the version-specific SAML assertion XML
1553 namespace URIs, as defined by the appropriate version-specific SAML "core"
1554 specification (see bibliography).

1555 With SAMLv2.0 specifically, the root element of the returned object may be either
1556 `<saml:Assertion>` or `<saml:EncryptedAssertion>`, where "saml" represents any XML
1557 namespace prefix that maps to the SAMLv2.0 assertion namespace URI:

1558 `urn:oasis:names:tc:SAML:2.0:assertion`

1559 File extension(s)

1560 None

1561 Macintosh File Type Code(s)

1562 None

1563 **Person & email address to contact for further information**

1564 This registration is made on behalf of the OASIS Security Services Technical Committee (SSTC)
1565 Please refer to the SSTC website for current information on committee chairperson(s) and their
1566 contact addresses: <http://www.oasis-open.org/committees/security/>. Committee members should
1567 submit comments and potential errata to the security-services@lists.oasis-open.org list. Others
1568 should submit them by filling out the web form located at [http://www.oasis-
1569 open.org/committees/comments/form.php?wg_abbrev=security](http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=security).

1570 Additionally, the SAML developer community email distribution list, [saml-dev@lists.oasis-
1571 open.org](mailto:saml-dev@lists.oasis-open.org), may be employed to discuss usage of the `application/samlassertion+xml`
1572 MIME media type. The "saml-dev" mailing list is publicly archived here: [http://lists.oasis-
1573 open.org/archives/saml-dev/](http://lists.oasis-open.org/archives/saml-dev/). To post to the "saml-dev" mailing list, one must subscribe to it. To
1574 subscribe, send a message with the single word "subscribe" in the message body, to: [saml-dev-
1575 request@lists.oasis-open.org](mailto:saml-dev-request@lists.oasis-open.org).

1576 **Intended usage**

1577 COMMON

1578 **Author/Change controller**

1579 The SAML specification sets are a work product of the OASIS Security Services Technical
1580 Committee (SSTC). OASIS and the SSTC have change control over the SAML specification sets.

1581 **Bibliography**

- 1582 [LAP] *"Liberty Alliance Project"*. See <http://www.projectliberty.org/>
- 1583 [OASIS] *"Organization for the Advancement of Structured Information Systems"*.
1584 See <http://www.oasis-open.org/>
- 1585 [RFC3023] M. Murata, S. St.Laurent, D. Kohn, "XML Media Types", IETF Request for
1586 Comments 3023, January 2001. Available as [http://www.rfc-
1587 editor.org/rfc/rfc3023.txt](http://www.rfc-editor.org/rfc/rfc3023.txt)
- 1588 [SAMLv1.0] OASIS Security Services Technical Committee, "Security Assertion
1589 Markup Language (SAML) Version 1.0 Specification Set". OASIS
1590 Standard 200205, November 2002. Available as [http://www.oasis-
1591 open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip](http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip)
- 1592 [SAMLv1Bind] Prateek Mishra et al., "Bindings and Profiles for the OASIS Security
1593 Assertion Markup Language (SAML)", OASIS, November 2002.
1594 Document ID oasis-sstc-saml-bindings-1.0. See [http://www.oasis-
1595 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1596 [SAMLv1Core] Phillip Hallam-Baker et al., "Assertions and Protocol for the OASIS
1597 Security Assertion Markup Language (SAML)", OASIS, November 2002.
1598 Document ID oasis-sstc-saml-core-1.0. See [http://www.oasis-
1599 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1600 [SAMLv1Sec] Chris McLaren et al., "Security Considerations for the OASIS Security
1601 Assertion Markup Language (SAML)", OASIS, November 2002.
1602 Document ID oasis-sstc-saml-sec-consider-1.0. See [http://www.oasis-
1603 open.org/committees/security/](http://www.oasis-open.org/committees/security/)
- 1604 [SAMLv1.1] OASIS Security Services Technical Committee, "Security Assertion
1605 Markup Language (SAML) Version 1.1 Specification Set". OASIS
1606 Standard 200308, August 2003. Available as [http://www.oasis-
1607 open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip](http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip)
- 1608 [SAMLv11Bind] E. Maler et al. "Bindings and Profiles for the OASIS Security Assertion
1609 Markup Language (SAML)". OASIS, September 2003. Document ID

1610		oasis-sstc-saml-bindings-1.1. http://www.oasis-open.org/committees/security/
1611		
1612	[SAMLv11Core]	E. Maler et al. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)". OASIS, September 2003. Document ID oasis-sstc-saml-core-1.1. http://www.oasis-open.org/committees/security/
1613		
1614		
1615	[SAMLv11Sec]	E. Maler et al. "Security Considerations for the OASIS Security Assertion Markup Language (SAML)". OASIS, September 2003. Document ID oasis-sstc-saml-sec-consider-1.1. http://www.oasis-open.org/committees/security/
1616		
1617		
1618		
1619	[SAMLv2.0]	OASIS Security Services Technical Committee, "Security Assertion Markup Language (SAML) Version 2.0 Specification Set". OASIS Standard, 15-Mar-2005. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip
1620		
1621		
1622		
1623	[SAMLv2Bind]	S. Cantor et al., "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-bindings-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
1624		
1625		
1626		
1627	[SAMLv2Core]	S. Cantor et al., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-core-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
1628		
1629		
1630		
1631	[SAMLv2Prof]	S. Cantor et al., "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-profiles-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf
1632		
1633		
1634		
1635	[SAMLv2Sec]	F. Hirsch et al., "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS, March 2005. Document ID saml-sec-consider-2.0-os. Available at: http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf
1636		
1637		
1638		
1639	[SSTC]	"OASIS Security Services Technical Committee". See http://www.oasis-open.org/committees/security/
1640		
1641	[XML]	Bray, T., Paoli, J., Sperberg-McQueen, C.M. and E. Maler, François Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml, Feb 2004, Available as http://www.w3.org/TR/REC-xml/
1642		
1643		
1644		

1645 Appendix B. Acknowledgments

1646 The editors would like to acknowledge the contributions of the OASIS Security Services Technical
1647 Committee, whose voting members at the time of publication were:

- 1648 • Conor Cahill, AOL
- 1649 • John Hughes, Atos Origin
- 1650 • Hal Lockhart, BEA Systems
- 1651 • Mike Beach, Boeing
- 1652 • Rebekah Metz, Booz Allen Hamilton
- 1653 • Rick Randall, Booz Allen Hamilton
- 1654 • Ronald Jacobson, Computer Associates
- 1655 • Gavenraj Sodhi, Computer Associates
- 1656 • Thomas Wisniewski, Entrust
- 1657 • Carolina Canales-Valenzuela, Ericsson
- 1658 • Dana Kaufman, Forum Systems
- 1659 • Irving Reid, Hewlett-Packard
- 1660 • Guy Denton, IBM
- 1661 • Heather Hinton, IBM
- 1662 • Maryann Hondo, IBM
- 1663 • Michael McIntosh, IBM
- 1664 • Anthony Nadalin, IBM
- 1665 • Nick Ragouzis, Individual
- 1666 • Scott Cantor, Internet2
- 1667 • Bob Morgan, Internet2
- 1668 • Peter Davis, Neustar
- 1669 • Jeff Hodges, Neustar
- 1670 • Frederick Hirsch, Nokia
- 1671 • Senthil Sengodan, Nokia
- 1672 • Abbie Barbir, Nortel Networks
- 1673 • Scott Kiester, Novell
- 1674 • Cameron Morris, Novell
- 1675 • Paul Madsen, NTT
- 1676 • Steve Anderson, OpenNetwork
- 1677 • Ari Kermaier, Oracle
- 1678 • Vamsi Motukuru, Oracle
- 1679 • Darren Platt, Ping Identity
- 1680 • Prateek Mishra, Principal Identity
- 1681 • Jim Lien, RSA Security
- 1682 • John Linn, RSA Security
- 1683 • Rob Philpott, RSA Security
- 1684 • Dipak Chopra, SAP
- 1685 • Jahan Moreh, Sigaba
- 1686 • Bhavna Bhatnagar, Sun Microsystems

- 1687 • Eve Maler, Sun Microsystems
- 1688 • Ronald Monzillo, Sun Microsystems
- 1689 • Emily Xu, Sun Microsystems
- 1690 • Greg Whitehead, Trustgenix

1691 The editors also would like to acknowledge the following former SSTC members for their contributions to
1692 this or previous versions of the OASIS Security Assertions Markup Language Standard:

- 1693 • Stephen Farrell, Baltimore Technologies
- 1694 • David Orchard, BEA Systems
- 1695 • Krishna Sankar, Cisco Systems
- 1696 • Zahid Ahmed, CommerceOne
- 1697 • Tim Alsop, CyberSafe Limited
- 1698 • Carlisle Adams, Entrust
- 1699 • Tim Moses, Entrust
- 1700 • Nigel Edwards, Hewlett-Packard
- 1701 • Joe Pato, Hewlett-Packard
- 1702 • Bob Blakley, IBM
- 1703 • Marlena Erdos, IBM
- 1704 • Marc Chanliau, Netegrity
- 1705 • Chris McLaren, Netegrity
- 1706 • Lynne Rosenthal, NIST
- 1707 • Mark Skall, NIST
- 1708 • Charles Knouse, Oblix
- 1709 • Simon Godik, Overxeer
- 1710 • Charles Norwood, SAIC
- 1711 • Evan Prodromou, Securant
- 1712 • Robert Griffin, RSA Security (former editor)
- 1713 • Sai Allarvarpu, Sun Microsystems
- 1714 • Gary Ellison, Sun Microsystems
- 1715 • Chris Ferris, Sun Microsystems
- 1716 • Mike Myers, Traceroute Security
- 1717 • Phillip Hallam-Baker, VeriSign (former editor)
- 1718 • James Vanderbeek, Vodafone
- 1719 • Mark O'Neill, Vordel
- 1720 • Tony Palmer, Vordel

1721 Finally, the editors wish to acknowledge the following people for their contributions of material used as
1722 input to the OASIS Security Assertions Markup Language specifications:

- 1723 • Thomas Gross, IBM
- 1724 • Birgit Pfitzmann, IBM

1725 **Appendix C. Notices**

1726 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
1727 might be claimed to pertain to the implementation or use of the technology described in this document or
1728 the extent to which any license under such rights might or might not be available; neither does it represent
1729 that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to
1730 rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made
1731 available for publication and any assurances of licenses to be made available, or the result of an attempt
1732 made to obtain a general license or permission for the use of such proprietary rights by implementors or
1733 users of this specification, can be obtained from the OASIS Executive Director.

1734 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or
1735 other proprietary rights which may cover technology that may be required to implement this specification.
1736 Please address the information to the OASIS Executive Director.

1737 **Copyright © OASIS Open 2005. All Rights Reserved.**

1738 This document and translations of it may be copied and furnished to others, and derivative works that
1739 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
1740 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
1741 this paragraph are included on all such copies and derivative works. However, this document itself may
1742 not be modified in any way, such as by removing the copyright notice or references to OASIS, except as
1743 needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights
1744 defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it
1745 into languages other than English.

1746 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
1747 or assigns.

1748 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1749 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
1750 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
1751 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.