



# Extensible Resource Identifier (XRI) Resolution V2.0

Working Draft 10, 18 March 2006

## Document identifier:

xri-resolution-V2.0-wd-10

## Location:

<http://docs.oasis-open.org/xri/xri/V2.0>

## Editors:

Gabe Wachob, Visa International <[gwachob@visa.com](mailto:gwachob@visa.com)>  
Drummond Reed, Cordance <[drummond.reed@cordance.net](mailto:drummond.reed@cordance.net)>  
Les Chasen, NeuStar <[les.chasen@neustar.biz](mailto:les.chasen@neustar.biz)>  
William Tan, NeuStar <[william.tan@neustar.biz](mailto:william.tan@neustar.biz)>  
Steve Churchill, XDI.ORG <[steven.churchill@xdi.org](mailto:steven.churchill@xdi.org)>

## Contributors:

Dave McAlpin, Epok <[dave.mcalpin@epok.net](mailto:dave.mcalpin@epok.net)>  
Chetan Sabnis, Epok <[chetan.sabnis@epok.net](mailto:chetan.sabnis@epok.net)>  
Peter Davis, Neustar <[peter.davis@neustar.biz](mailto:peter.davis@neustar.biz)>  
Victor Grey, PlaNetwork <[victor.grey@planetnetwork.org](mailto:victor.grey@planetnetwork.org)>  
Mike Lindelsee, Visa International <[mlindels@visa.com](mailto:mlindels@visa.com)>

## Abstract:

This document defines both generic and trusted HTTP(S)-based resolution protocols for Extensible Resource Identifiers (XRIs) as defined by *Extensible Resource Identifier (XRI) Syntax V2.0 [XRISyntax]* or higher. For the set of XRIs defined to provide identifier metadata, see *Extensible Resource Identifier (XRI) Metadata V2.0 [XRIMetadata]*. For a basic introduction to XRIs, see the *XRI 2.0 FAQ [XRIFAQ]*.

## Status:

This document was last revised or approved by the XRI Technical Committee on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/xri>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/xri/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/xri>.

---

## 42 Table of Contents

43	1	Introduction .....	5
44	1.1	Overview of XRI Resolution Architecture.....	5
45	1.2	Structure of this Specification.....	7
46	1.3	Examples of XRI Resolution Requests and Responses.....	8
47	1.4	Terminology and Notation.....	8
48	2	Namespaces .....	9
49	2.1	XRI Namespaces for XRI Resolution.....	9
50	2.1.1	XRIs Reserved for XRI Resolution .....	9
51	2.1.2	XRIs Assigned to XRI Resolution Service Types .....	9
52	2.2	XML Namespaces for XRI Resolution.....	9
53	2.3	Media Types for XRI Resolution .....	10
54	3	XRDS Documents.....	11
55	3.1	XRDS and XRD Namespaces.....	11
56	3.2	XRD Elements and Attributes .....	11
57	3.2.1	Management Elements.....	13
58	3.2.2	Authority Trust Elements .....	13
59	3.2.3	Synonym Elements.....	14
60	3.2.4	Service Endpoint Elements.....	14
61	3.2.5	Service Endpoint Trust Elements .....	15
62	3.2.6	Service Endpoint Selection Elements.....	15
63	3.3	XRD Attribute Processing Rules .....	16
64	3.3.1	ID Attribute .....	16
65	3.3.2	Version Attribute .....	16
66	3.3.3	Priority Attribute .....	16
67	3.4	XRI Encoding Requirements.....	17
68	4	Inputs and Outputs.....	18
69	4.1	Inputs.....	18
70	4.1.1	QXRI (Authority String, Path String, and Query String).....	19
71	4.1.2	Resolution Media Type .....	19
72	4.1.3	Service Type.....	20
73	4.1.4	Service Media Type .....	20
74	4.2	Outputs.....	21
75	4.2.1	XRDS Document.....	21
76	4.2.2	XRD Document.....	21
77	4.2.3	URI List .....	22
78	4.2.4	HTTP(S) Redirect .....	22
79	5	Generic Authority Resolution .....	23
80	5.1	XRI Authority Resolution .....	23
81	5.1.1	Service Type and Service Media Type.....	23
82	5.1.2	Protocol.....	24
83	5.1.3	Community Root Authorities .....	26
84	5.1.4	Qualified Subsegments.....	26

85	5.1.5 Cross-References .....	27
86	5.1.6 Recursing Authority Resolution .....	27
87	5.1.7 Construction of the Next Authority URI.....	28
88	5.2 IRI Authority Resolution .....	28
89	5.2.1 Service Type and Media Type .....	29
90	5.2.2 Protocol.....	29
91	5.2.3 Optional Use of HTTPS .....	29
92	6 Trusted Authority Resolution.....	30
93	6.1 HTTPS.....	30
94	6.1.1 Service Type and Service Media Type .....	30
95	6.1.2 Protocol.....	30
96	6.2 SAML.....	30
97	6.2.1 Service Type and Service Media Type .....	31
98	6.2.2 Protocol.....	31
99	6.2.3 Recursing Authority Resolution .....	32
100	6.2.4 Client Validation of XRDS .....	32
101	6.2.5 Correlation of ProviderID and KeyInfo Elements.....	33
102	6.3 HTTPS+SAML.....	34
103	6.3.1 Service Type and Service Media Type .....	34
104	6.3.2 Protocol.....	34
105	7 Proxy Resolution.....	35
106	7.1 Service Type and Media Types.....	35
107	7.2 HXRIs .....	36
108	7.3 QXRI Query Parameters .....	36
109	7.4 HTTP(S) Accept Headers .....	37
110	7.5 Null Resolution Media Type .....	38
111	7.6 Outputs and HTTP(S) Redirects .....	38
112	7.7 Differences Between Proxy Resolution Servers .....	38
113	8 Service Endpoint Selection .....	39
114	8.1 Processing Rules .....	39
115	8.2 Matching Rules.....	40
116	8.2.1 Match Attribute Values.....	40
117	8.2.2 Service Type Matching .....	41
118	8.2.3 Service Media Type Matching .....	41
119	8.2.4 Path String Matching.....	41
120	8.3 Selection Rules .....	42
121	8.4 Construction of Service Endpoint URIs.....	42
122	9 Reference Processing.....	44
123	9.1 Synonyms.....	44
124	9.2 Processing Rules .....	45
125	9.3 Nesting XRDS Documents.....	46
126	10 Error Processing .....	48
127	10.1 Error Codes .....	48
128	10.2 Error Context Strings.....	50

129	10.3 Error Handling in Recursing and Proxy Resolution.....	50
130	11 Use of HTTP(S) .....	51
131	11.1 HTTP Errors .....	51
132	11.2 HTTP Headers .....	51
133	11.2.1 Caching.....	51
134	11.2.2 Location .....	51
135	11.2.3 Content-Type .....	51
136	11.3 Other HTTP Features.....	51
137	11.4 Caching and Efficiency.....	52
138	12 Extensibility and Versioning.....	53
139	12.1 Extensibility .....	53
140	12.1.1 Extensibility of XRDS.....	53
141	12.1.2 Other Points of Extensibility.....	53
142	12.2 Versioning .....	54
143	12.2.1 Version Numbering .....	54
144	12.2.2 Versioning of the XRI Resolution Specification .....	54
145	12.2.3 Versioning of XRDS.....	54
146	12.2.4 Versioning of Protocols.....	55
147	13 Security and Data Protection .....	56
148	13.1 DNS Spoofing or Poisoning .....	56
149	13.2 HTTP Security .....	56
150	13.3 SAML Considerations .....	56
151	13.4 Limitations of Trusted Resolution.....	56
152	13.5 Community Root Authorities .....	57
153	13.6 Caching Authorities .....	57
154	13.7 Recursing and Proxy Resolution.....	57
155	13.8 Denial-Of-Service Attacks .....	57
156	14 References.....	58
157	14.1 Normative .....	58
158	14.2 Informative.....	59
159	Appendix A. XML Schema for XRDS and XRD (Normative).....	60
160	Appendix B. RelaxNG Compact Syntax Schema for XRDS and XRD (Informative) .....	63
161	Appendix C. Media Type Definition for application/xrds+xml (Normative) .....	64
162	Appendix D. Media Type Definition for application/xrd+xml (Normative).....	65
163	Appendix E. Example Local Resolver Interface Definition (Informative).....	66
164	Appendix F. Examples of Generic Authority Resolution (Informative) .....	70
165	Appendix G. Examples of SAML Trusted Authority Resolution (Informative) .....	71
166	Appendix H. Examples of Service Endpoint Selection (Informative).....	72
167	Appendix I. Acknowledgments .....	73
168	Appendix J. Notices .....	74
169		

170

# 1 Introduction

171 Extensible Resource Identifier (XRI) provides a uniform syntax for abstract structured identifiers  
 172 as defined in **[XRISyntax]**. Because XRIs may be used across a wide variety of communities and  
 173 applications (as Web addresses, messaging addresses, database keys, filenames, directory  
 174 keys, object IDs, XML IDs, tags, etc.), no single resolution mechanism may prove appropriate for  
 175 all XRIs. However, in the interest of promoting interoperability, this specification defines a  
 176 standard protocol for resolving XRIs using HTTP(S). Both generic and trusted versions are  
 177 defined (the latter using HTTPS **[RFC2818]** and/or signed SAML assertions **[SAML]**). In addition,  
 178 an HTTP(S) proxy resolution version is specified both to provide network-based resolution  
 179 services and for backwards compatibility with existing HTTP(S) infrastructure.

180

## 1.1 Overview of XRI Resolution Architecture

181 Resolution is the function of dereferencing an identifier to a set of data and metadata describing  
 182 the identified resource. For example, in DNS, a domain name is typically resolved using the UDP  
 183 protocol into the IP address or other attributes of an Internet host. A federated domain name such  
 184 as `docs.oasis-open.org` is resolved recursively from right to left, i.e., first the resolver queries  
 185 the `org` nameserver for the IP address of the name-server for `oasis-open`, then it queries the  
 186 `oasis-open` nameserver for the IP address for `docs`.

187 Non-recursing resolvers rely on *recursing nameservers* to do this work. For example, a non-  
 188 recursing resolver might query a recursing nameserver for the entire DNS name `docs.oasis-  
 189 open.org`. The nameserver would then do the job of querying the `org` nameserver for the IP  
 190 address of `oasis-open`, then the `oasis-open` nameserver or the IP address of `docs`, and  
 191 lastly return the result to the resolver. A recursing nameserver typically caches all these resource  
 192 records so it can answer subsequent queries directly from cache.

193 XRI resolution follows this same architecture except at a higher level of abstraction, i.e., rather  
 194 than resolving a domain name into an attribute of a text resource descriptor using UDP, it  
 195 resolves an XRI into a XML resource descriptor using HTTP(S). Table 1 provides an overview of  
 196 the comparison between DNS and XRI resolution architectures.

Resolution Component	DNS Architecture	XRI Architecture
Identifier	domain name	XRI (authority + path + query)
Resource record format	text (resource record)	XML (XRDS document)
Attribute identifier	string	anyURI
Network endpoint identifier	IP address	URI
Synonyms	CNAME	Local, Canonical, External
Primary resolution protocol	UDP	HTTP(S)
Trusted resolution options	DNSSEC	HTTPS and/or SAML
Resolution client	local resolver	local resolver
Resolution server	authoritative nameserver	authority resolution service
Recursing resolution server	recursing nameserver	recursing authority resolution service
Proxy resolution	--	proxy resolver

197

**Table 1: Comparing DNS and XRI resolution architecture.**

198 As Table 1 notes, XRI resolution architecture adds one more component to the set used by  
 199 DNS—a *proxy resolver*. A proxy resolver is simply a local XRI resolver with an HTTP(S) interface.  
 200 Proxy resolvers enable applications—even those that do not natively understand XRIs but can  
 201 process HTTP URIs—to access the functions of an XRI resolver remotely.  
 202 Figure 1 shows four typical scenarios of how these components can interact to resolve  
 203 `xri://(example.root)*foo*bar` (note that unlike DNS, this works from left-to-right).

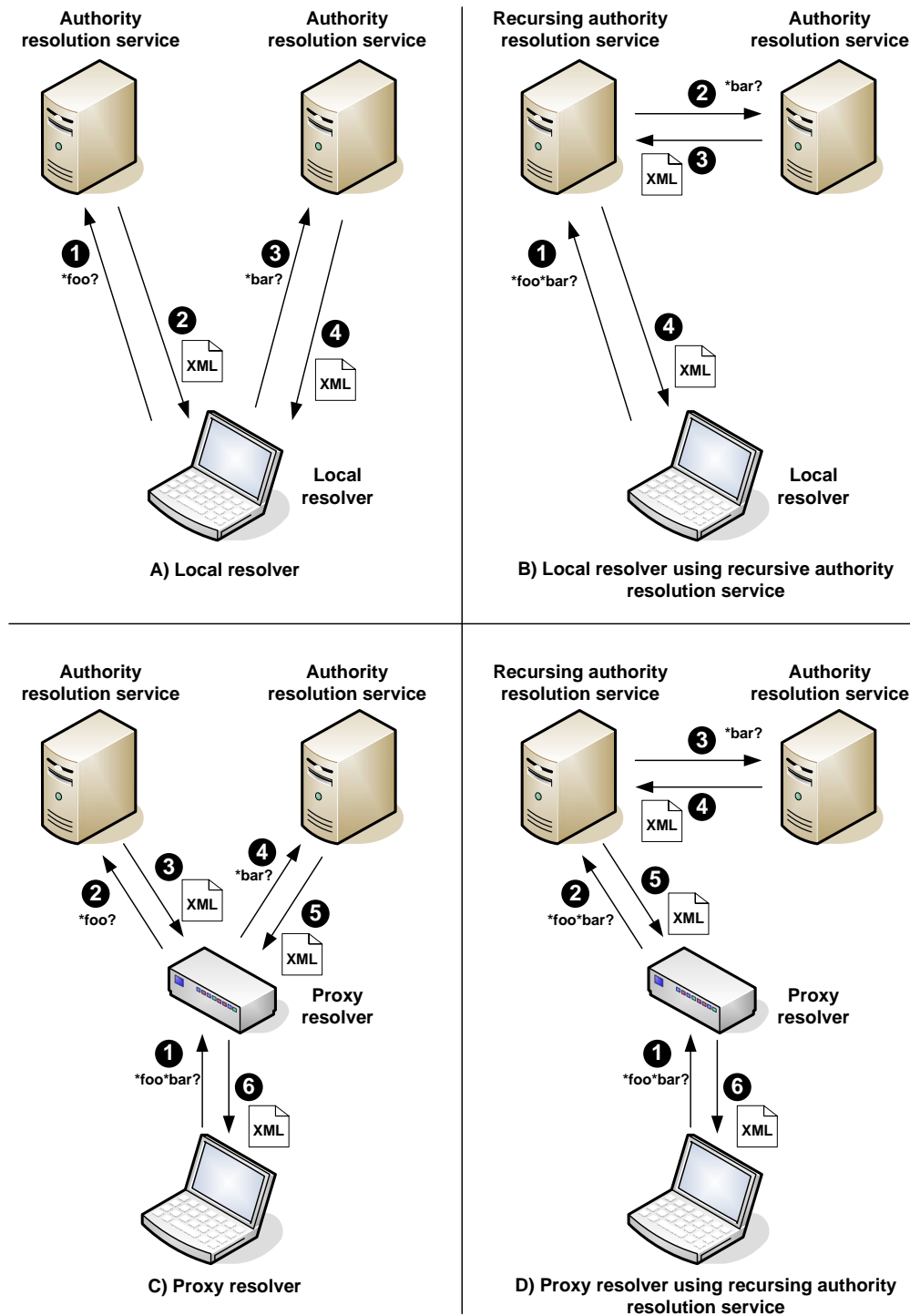


Figure 1: Four typical scenarios for XRI authority resolution.

204  
205

206 In each of these scenarios, two phases of XRI resolution can be involved:

- 207 • *Phase 1: authority resolution.* This is the phase required to resolve the authority segment  
208 of an XRI into the an XRDS document describing the target authority. Authority resolution  
209 works recursively from left-to-right across each subsegment in the authority segment of  
210 the XRI (subsegments are delimited using a "\*" or "!" character). For example, in the XRI  
211 `xri://(example.root)*foo*bar`, the authority subsegments are `(example.root)`  
212 (the community root authority, in this case expressed as an cross-reference), `*foo`, (the  
213 first resolvable subsegment), and `*bar`, (the second resolvable subsegment). Note that a  
214 resolver must be preconfigured (or have its own way of discovering) the community root  
215 authority starting point, so the community root subsegment is never resolved.
- 216 • *Phase 2: service endpoint selection.* Once authority resolution is complete, the optional  
217 second phase of XRI resolution is to select specific metadata from the final XRDS  
218 document retrieved. Although an XRDS document may contain any type of metadata  
219 describing the target resource, this specification defines a ruleset for selecting *service*  
220 *endpoints*: descriptors of concrete URIs at which network services are available for the  
221 target resource. An XRI resolver may optionally use the path and query components of  
222 an XRI to select the service endpoint(s) to return to a calling application.

223 It is worth highlighting several other key differences between DNS and XRI resolution:

- 224 • *HTTP.* As a resolution protocol, HTTP not only makes it easy to deploy XRI resolution  
225 services (including proxy resolution services), but can employ both HTTP security  
226 standards (e.g., HTTPS) and XML-based security standards (e.g., SAML). Although less  
227 efficient than UDP, HTTP(S) is suitable for the higher level of abstraction represented by  
228 XRIs and can take advantage of the full caching capabilities of modern web servers.
- 229 • *XRDS documents.* This simple, extensible XML resource description format makes it  
230 easy to describe the capabilities of any XRI-identified resource in a manner that can be  
231 consumed by any XML-aware application.
- 232 • *Synonyms and references.* DNS uses the CNAME attribute to establish equivalence  
233 between domain names. The XRI resolution format includes three XRI mapping elements  
234 (local IDs, canonical IDs, and external references) to provide robust support for mapping  
235 XRIs to other XRIs, IRIs, or URIs that represent the same resource. This is particularly  
236 useful for discovering and mapping persistent identifiers often required by trust  
237 infrastructures. The use of XRI references also enables multiple authorities to maintain  
238 distributed XRDS documents describing the same logical resource.
- 239 • *Service endpoint descriptors.* DNS can use NAPTR records to do string transformations  
240 into URIs representing network endpoints. XRDS documents have *service endpoint*  
241 *descriptors*—elements that describe the set of URIs at which a particular type of service  
242 is available. Each service endpoint may present a different subset, type, or  
243 representation of data or metadata for the identified resource. Thus XRI resolution can  
244 serve as a lightweight, interoperable discovery mechanism for resource attributes  
245 available via LDAP, UDDI, or other directory or discovery protocols.

## 246 1.2 Structure of this Specification

247 This specification is structured into the following major sections:

- 248 • *Namespaces* (section 2) specifies the XRI and XML namespaces and media types used  
249 for the XRI resolution protocol.
- 250 • *XRDS Documents* (section 3) specifies a simple, flexible XML-based container for XRI  
251 resolution metadata or other metadata describing a resource.
- 252 • *Inputs and Outputs* (section 4) specifies the standard input parameters and output  
253 formats for XRI resolution.

- 254 • *Generic Authority Resolution* (section 5) specifies a simple resolution protocol for the  
255 authority segment of an XRI using HTTP/HTTPS as a transport.
- 256 • *Trusted Authority Resolution* (section 6) specifies three extensions to generic authority  
257 resolution for creating a chain of trust between the participating identifier authorities using  
258 HTTPS connections, SAML assertions, or both.
- 259 • *Proxy Resolution* (section 7) specifies an HTTP(S) interface for an XRI resolver plus a  
260 format for expressing an XRI as an HTTP(S) URI to provide backwards compatibility with  
261 existing HTTP(S) infrastructure.
- 262 • *Service Endpoint Selection* (section 8) specifies an optional second phase of resolution  
263 for selecting a set of service endpoints from an XRDS document.
- 264 • *Reference Processing* (section 9) specifies how a resolver follows XRI references to  
265 enable federation of XRDS documents across multiple XRI authorities.
- 266 • *Error Processing* (section 10) specifies error codes and error handling.
- 267 • *Use of HTTP(S)* (section 11) specifies how the XRI resolution protocol leverages features  
268 of the HTTP(S) protocol.
- 269 • *Extensibility and Versioning* (section 12) describes how the XRI resolution protocol can  
270 be easily extended and how new versions will be identified and accommodated.
- 271 • *Security and Data Protection* (section 13) summarizes key security and privacy  
272 considerations for XRI resolution infrastructure.

## 273 1.3 Examples of XRI Resolution Requests and Responses

274 To minimize non-normative material in the main body of the specification, extensive examples of  
275 XRI resolution requests and responses are compiled in Appendix F for generic resolution,  
276 Appendix G for SAML trusted resolution, and Appendix H for service endpoint selection.

## 277 1.4 Terminology and Notation

278 The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”,  
279 “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this  
280 document are to be interpreted as described in [RFC2119]. When these words are not capitalized  
281 in this document, they are meant in their natural language sense.

282 This specification uses the Augmented Backus-Naur Form (ABNF) syntax notation defined in  
283 [RFC2234].

284 Other terms used in this document and not defined herein are defined in the glossary in Appendix  
285 C of [XRISyntax].

286 Formatting conventions used in this document:

287 `Examples look like this.`

288 `ABNF productions look like this.`

289 In running text, XML elements, attributes, and values look like this.



---

## 290 2 Namespaces

### 291 2.1 XRI Namespaces for XRI Resolution

292 As defined in section 2.2.1.2 of [XRISyntax], the GCS symbol "\$" is reserved for special  
293 identifiers assigned by XRI TC specifications, other OASIS specifications, or other standards  
294 bodies. (See also [XRIMetadata].) This section specifies the \$ namespaces reserved for XRI  
295 resolution.

#### 296 2.1.1 XRIs Reserved for XRI Resolution

297 The XRIs in Table 2 are assigned by this specification for the purposes of XRI resolution and  
298 resource description.

XRI	Use	See Section
xri://\$res	Namespace for XRI resolution service types	2.1.2
xri://\$xrds	Namespace for the generic XRDS (Extensible Resource Descriptor Sequence) schema (not versioned)	2.2
xri://\$xrd	Namespace for the XRD (Extensible Resource Descriptor) schema	2.2
xri://\$xrd*(\$v*2.0)	Version 2.0 of above (using XRI version metadata as defined in [XRIMetadata])	2.2

299 **Table 2: XRIs reserved for XRI resolution.**

#### 300 2.1.2 XRIs Assigned to XRI Resolution Service Types

301 The XRIs in Table 3 are assigned to the XRI resolution service types defined in this specification.

XRI	Use	See Section
xri://\$res*auth	Authority resolution service	5
xri://\$res*auth*(\$v*2.0)	Version 2.0 of above	5
xri://\$res*proxy	HTTP(S) proxy resolution service	7
xri://\$res*proxy*(\$v*2.0)	Version 2.0 of above	7

302 **Table 3: XRIs assigned to identify XRI resolution service types.**

303 Using the standard XRI extensibility mechanisms described in [XRISyntax], the "\$res"  
304 namespace may be extended by other authorities besides the XRI Technical Committee. See  
305 [XRIMetadata] for more information about extending "\$" namespaces.

### 306 2.2 XML Namespaces for XRI Resolution

307 Throughout this document, the following XML namespaces prefixes have the meanings defined in  
308 Table 4 whether or not they are explicitly declared in the example or text.

Prefix	XML Namespace	Reference
xs	http://www.w3.org/2001/XMLSchema	[XMLSchema]
saml	urn:oasis:names:tc:SAML:2.0:assertion	[SAML]
ds	http://www.w3.org/2000/09/xmldsig#	[XMLDSig]
xrds	xri://\$xrds	Section 2.1.1 of this document
xrd	xri://\$xrd*(\$v*2.0)	Section 2.1.1 of this document

309

**Table 4: XML namespace prefixes used in this specification.**

## 310 2.3 Media Types for XRI Resolution

311 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as  
 312 defined by [RFC2046], particularly in HTTP Accept headers as specified in [RFC2616]. Table 5  
 313 specifies the media types used for XRI resolution.

Media Type	Usage	Reference
application/xrds+xml	Content type for returning the full XRDS document describing a resolution chain	Appendix C
application/xrd+xml	Content type for returning only the final XRD descriptor in a resolution chain	Appendix D
text/uri-list	Content type for returning a list of URIs output from the service endpoint selection process defined in section 8	Section 5 of [RFC2483]

314

**Table 5: Media types defined or used in this specification.**

315 To provide full control of XRI resolution via an HTTP interface, these media types accept media  
 316 types parameters as defined in Table 6.

Parameter	Values	Applies to Media Type	Usage
trust	none https saml https+saml	application/xrds+xml application/xrd+xml text/uri-list	Specifies use of a trusted resolution protocol (section 6)
refs	true false	application/xrds+xml application/xrd+xml text/uri-list	Specifies whether references should be followed during resolution (section 9)
sep	true false	application/xrds+xml application/xrd+xml	Specifies whether service endpoint selection should be performed (section 8)

317

**Table 6: Parameters for the media types defined in Table 5.**

318 See sections 4 - 8 for more about usage of these media types and parameters.

---

## 319 **3 XRDS Documents**

320 XRI resolution uses a simple, extensible XML format called an XRDS (Extensible Resource  
321 Descriptor Sequence) document. An XRDS document contains one or more XRD (Extensible  
322 Resource Descriptor) documents. While this specification defines only the XRD elements  
323 necessary to support XRI resolution, XRD documents can easily be extended to publish any form  
324 of metadata about the resources they describe.

### 325 **3.1 XRDS and XRD Namespaces**

326 An XRDS document is intended to serve exclusively as an XML container document for XML  
327 schemas from other XML namespaces. Therefore it has only a single root element `xrds:XRDS` in  
328 its own XML namespace identified by the XRI `xri://$xrds`. It also has a single attribute,  
329 `xrds:XRDS/@xrds:ref` of type `anyURI` that identifies the resource described by the XRDS  
330 document. The formal XML schema definition of an XRDS document is provided in Appendix A.

331 The elements in the XRD schema are intended for generic resource description, including the  
332 metadata necessary for XRI resolution. Since the XRD schema has simple semantics that may  
333 evolve over time, the version defined in this specification uses the XML namespace  
334 `xri://$xrd*($v*2.0)`. This namespace is versioned using XRI version metadata as defined  
335 in **[XRIMetadata]**.

336 This namespace architecture enables the XRDS namespace to remain constant while the XRD  
337 namespace (and the namespaces of other XML elements that may be included in an XRDS  
338 document) may be versioned over time. See section 12.2 for more about versioning of the XRD  
339 schema.

### 340 **3.2 XRD Elements and Attributes**

341 The following example XRDS instance document illustrates the elements and attributes defined in  
342 the XRD schema:

```

343 <XRDS xmlns="xri://$xrds" ref="xri://(example.root)*foo">
344   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
345     <Query>*foo</Query>
346     <Status code="100"/>
347     <Expires>2005-05-30T09:30:10Z</Expires>
348     <ProviderID>urn:uuid:c9f812f3-6544-4e3c-874e-
349 d3ae79f4ef7b</ProviderID>
350     <LocalID>*baz</LocalID>
351     <CanonicalID>xri://(example.root)!1234!5678</CanonicalID>
352     <Ref>xri://!!4A76!C2F7!9033</Ref>
353     <Service>
354       <ProviderID>xri://!!1000!1234.5678</ProviderID>
355       <Type>xri://$res*auth*($v*2.0)</Type>
356       <MediaType>application/xrds+xml</MediaType>
357       <URI priority="10">http://resolve.example.com</URI>
358       <URI priority="15">http://resolve2.example.com</URI>
359       <URI>https://resolve.example.com</URI>
360     </Service>
361     <Service>
362       <ProviderID>xri://!!1000!1234.5678</ProviderID>
363       <Type>xri://$res*auth*($v*2.0)</Type>
364       <MediaType>application/xrds+xml;trust=https</MediaType>
365       <URI>https://resolve.example.com</URI>
366     </Service>
367     <Service>
368       <Type match="null" />
369       <Path select="true">media/pictures</Path>
370       <MediaType select="true">image/jpeg</MediaType>
371       <URI append="path" >http://pictures.example.com</URI>
372     </Service>
373     <Service>
374       <Type match="null" />
375       <Path select="true">media/videos</Path>
376       <MediaType select="true">video/mpeg</MediaType>
377       <URI append="path" >http://videos.example.com</URI>
378     </Service>
379     <Service>
380       <ProviderID> xri://!!1000!1234.5678</ProviderID>
381       <Type match="null" />
382       <Path match="default" />
383       <URI>http://example.com/local</URI>
384     </Service>
385     <Service>
386       <Type>http://example.com/some/service/v3.1</Type>
387       <URI>http://example.com/some/service/endpoint</URI>
388     </Service>
389   </XRD>
390 </XRDS>

```

391 The normative XML schema definition of the XRD schema is provided in Appendix A. Additional  
392 normative requirements that cannot be captured in XML schema notation are specified in the  
393 following sections. In the case of any conflict, the normative text in this section shall prevail.

## 394 3.2.1 Management Elements

395 The first set of elements are used to manage XRDs, especially from the perspective of caching  
396 and error handling.

### 397 **xrd:XRD**

398 Container element for all other XRD elements. Includes an optional `xml:id` attribute of  
399 type `xs:ID`. This attribute is REQUIRED in trusted resolution to uniquely identify this  
400 element within the containing `xrd:XRDS` document. It also includes an optional  
401 `xrd:idref` attribute of type `xs:idref`. This attribute is REQUIRED in trusted resolution  
402 when an XRD element in a nested `xrd:XRDS` document must reference a previously  
403 included XRD instance. See sections 3.3 and 9.2. Lastly, it includes an `xrd:version`  
404 attribute that is optional for uses outside of XRI resolution but REQUIRED for XRI  
405 resolution as defined in section 3.3.2

### 406 **xrd:XRD/xrd:Query**

407 0 or 1 per `xrd:XRD` element. Expresses the XRI, IRI, or URI reference in URI normal  
408 form whose resolution results in this `xrd:XRD` element. For XRI authority resolution, this  
409 must be a qualified subsegment of the authority component of the query XRI.

### 410 **xrd:XRD/xrd:Status**

411 0 or 1 per `xrd:XRD` element. Contains a required attribute `xrd:code` of type `xs:int`  
412 that provides a numeric status code. The contents of the element are an optional human-  
413 readable message string describing the status of the response. For XRI resolution,  
414 values of the Status element and `xrd:code` attribute are defined in section 10.

### 415 **xrd:XRD/xrd:Expires**

416 0 or 1 per `xrd:XRD` element. The date/time, in the form of `xs:dateTime`, after which  
417 this XRD cannot be relied upon. To promote interoperability, this date/time value  
418 SHOULD use the UTC "Z" time zone and SHOULD NOT use fractional seconds. A  
419 resolver using this XRD MUST NOT use the XRD after the time stated here. A resolver  
420 MAY discard this XRD before the time indicated in this result. If the HTTP transport  
421 caching semantics specify an expiry time earlier than the time expressed in this attribute,  
422 then a resolver MUST NOT use this XRD after the expiry time declared in the HTTP  
423 headers per section 13.2 of [RFC2616]. See section 11.2.1.

## 424 3.2.2 Authority Trust Elements

425 The second set of elements are for applications where trust must be established in the authority  
426 providing the XRD. These elements are OPTIONAL for generic authority resolution (section 5),  
427 but REQUIRED for trusted authority resolution (section 6).

### 428 **xrd:XRD/xrd:ProviderID**

429 0 or 1 per `xrd:XRD`. A unique identifier of type `xs:anyURI` for the authority producing  
430 this XRD. There MUST be negligible probability that the value of this element will be  
431 assigned as an identifier to any other authority. This element MUST be a persistent  
432 identifier such as a URN [RFC2141] or a fully persistent XRI [XRISyntax]. Note that for  
433 XRI authority resolution, the authority identified by this element is the *describing* authority  
434 (the producer of the current XRD), not the authority *described* by the XRD. The latter is  
435 specified in the `xrd:XRD/xrd:Service/xrd:ProviderID` element for a resolution  
436 service endpoint (see below).

### 437 **xrd:XRD/saml:Assertion**

438 0 or 1 per `xrd:XRD`. A SAML assertion from the *describing* authority (the one producing  
439 the current XRD) that asserts that the information contained in the current XRD is  
440 authoritative. Because the assertion is digitally signed and the digital signature

441 encompasses the containing `xrd:XRD` element, it also provides a mechanism for the  
442 recipient to detect unauthorized changes since the time the XRD was published.

443 Note that while a `saml:Issuer` element is required within a `saml:Assertion` element,  
444 this specification makes no requirement as to the value of the `saml:Issuer` element. It  
445 is up to the XRI community resolution root to place restrictions, if any, on the  
446 `saml:Issuer` element. A suitable approach is to use an XRI in URI-normal form that  
447 describes the community root authority. See section 5.1.3.

### 448 3.2.3 Synonym Elements

449 A third set of elements are used to provide *synonyms*—identifiers that are not character-for-  
450 character equivalent to the contents of the `Query` element, but which identify the same target  
451 resource. For more information about synonyms in XRI resolution, see section 9.1. Because there  
452 may be multiple instances of each type of synonym, all three of these elements have the global  
453 `xrd:priority` attribute (see section 3.3.3).

#### 454 **xrd:XRD/xrd:LocalID**

455 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST NOT be an absolute  
456 identifier. MUST be an interchangeable synonym for the contents of the  
457 `xrd:XRD/xrd:Query` element, i.e., an XRI, IRI, or URI reference in URI normal form  
458 assigned to the same target resource by the same authority producing the current XRD. It  
459 MUST resolve to the current XRD (with the exception of the `xrd:Query`, `xrd:Expires`,  
460 and `xrd:LocalID` elements.)

#### 461 **xrd:XRD/xrd:CanonicalID**

462 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier.  
463 MUST contain an absolute XRI, IRI, or URI that serves as a canonical identifier for the  
464 described resource, i.e. the preferred synonym among all synonyms. If the contents is an  
465 XRI, it may or may not be resolvable and if resolvable may or may not resolve to the  
466 same XRD. An XRD element SHOULD contain only one instance of a `CanonicalID`  
467 element, however in some circumstances (such as the merging of two previously distinct  
468 resources) this may not be possible. Applications using this element to identify the target  
469 resource SHOULD try alternate instances if the top priority instance is not recognized.

#### 470 **xrd:XRD/xrd:Ref**

471 0 or more per `xrd:XRD` element. MUST be an absolute identifier. Identical to  
472 `xrd:XRD/xrd:CanonicalID` except this identifier MUST be assigned by a different  
473 authority than the authority producing the current XRD. Resolution of this reference MAY  
474 produce a different XRD for the described resource. See section 9 for complete details of  
475 reference processing.

### 476 3.2.4 Service Endpoint Elements

477 The next set of elements are used to describe service endpoints—the set of network endpoints  
478 advertised in an XRD for performing further resolution, obtaining further metadata, or interacting  
479 directly with the described resource. Again, because there can be more than one instance of a  
480 particular type of service endpoint that satisfies a service endpoint selection query, or more than  
481 one instance of a service endpoint URI, these elements both have the global `xrd:priority`  
482 attribute (see section 3.3.3).

#### 483 **xrd:XRD/xrd:Service**

484 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.

#### 485 **xrd:XRD/xrd:Service/xrd:URI**

486 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:anyURI`. If present, it  
487 indicates a transport-level URI for access to the capability described by the parent  
488 Service element. For the XRI resolution service types defined in section 2.1.2, this URI  
489 MUST be an HTTP or HTTPS URI. Other services may use other transport protocols. It  
490 includes an `xrd:append` attribute that governs construction of the final service endpoint  
491 URI. See section 8.4.

### 492 3.2.5 Service Endpoint Trust Elements

493 Similar to the authority trust elements, these elements enable trust to be established in the  
494 provider of the service endpoint. These elements are OPTIONAL for generic authority resolution  
495 (section 5), but REQUIRED for trusted authority resolution (section 6).

#### 496 **`xrd:XRD/xrd:Service/xrd:ProviderID`**

497 0 or 1 per `xrd:XRD/xrd:Service` element. Identical to the  
498 `xrd:XRD/xrd:ProviderID` above, except this identifies the provider of the described  
499 service endpoint instead of the provider of the current XRD. In XRI trusted resolution,  
500 when a resolution request is made to the authority at this service endpoint, the contents  
501 of the `xrd:XRD/xrd:ProviderID` element in the response MUST match the content of  
502 this element for correlation. See section 6.2.5. The same usage MAY apply to other  
503 services not defined in this specification.

#### 504 **`xrd:XRD/xrd:Service/ds:KeyInfo`**

505 0 or 1 per `xrd:XRD/xrd:Service` element. Provides the digital signature metadata  
506 necessary to validate an XRD provided as a resolution response by the described  
507 authority. This element comprises the key distribution method for trusted authority  
508 resolution in the XRI resolution framework—see section 6.2.5.

### 509 3.2.6 Service Endpoint Selection Elements

510 The final set of elements are used in XRI resolution to select service endpoints for further  
511 processing. They include two global attributes used for this purpose: `xrd:match` and  
512 `xrd:select`. See sections 8.2 and 8.3.

#### 513 **`xrd:XRD/xrd:Service/xrd:Type`**

514 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI`  
515 that identifies the type of capability available at this service endpoint. See section 2.1.2  
516 for the resolution service types defined in this specification.

#### 517 **`xrd:XRD/xrd:Service/xrd:MediaType`**

518 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of  
519 content available at this service endpoint. The value of this element MUST be of the form  
520 of a media type defined in [RFC2046]. See section 2.3 for the media types used in XRI  
521 resolution.

#### 522 **`xrd:XRD/xrd:Service/xrd:Path`**

523 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string  
524 value meeting the xri-path production defined in section 2.2.3 of [XRISyntax].

525 The XRD schema (Appendix A) allows other elements and attributes from other namespaces to  
526 be added throughout. As described in section 12.1.1, these points of extensibility can be used to  
527 deploy new XRI resolution schemes, new service description schemes, or other metadata about  
528 the described resource.

## 529 3.3 XRD Attribute Processing Rules

### 530 3.3.1 ID Attribute

531 For uses such as XRI trusted resolution (section 6.2) that require unique identification of multiple  
532 XRD elements within an XRDS document, the XRD element uses an optional `xml:id` attribute  
533 as defined by the W3C XML ID specification [XMLID]. If present, the value of this element MUST  
534 be unique for all elements in the containing XML document. Because an XRI resolver may need  
535 to assemble multiple XRDs received from different authority resolution services into one XRDS  
536 document, there MUST be negligible probability that the value of the `xrd:XRD/@xml:id`  
537 attribute is not globally unique. For this reason the value of this attribute SHOULD be a UUID as  
538 defined by [UUID] prefixed by a single underscore character "\_" in order to make it a legal  
539 *NCName* as required by [XMLID]. However the value of this attribute MAY be generated by any  
540 algorithm that fulfills the same requirements of global uniqueness and *NCName* conformance.

541 Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their  
542 XML document order MUST match the order in which they were resolved—see section 5.1.2.  
543 Also, if reference processing requires the same XRD to be included in an XRDS document twice  
544 (via a nested XRDS document), that XRD MUST reference the previous instance using the  
545 `xrd:XRD/@xml:idref` attribute as defined in section 9.

### 546 3.3.2 Version Attribute

547 Unlike the XRDS element, which is not intended to be versioned, the `xrd:XRD` element has the  
548 optional attribute `xrd:XRD/@xrd:version`. Use of this attribute is REQUIRED for XRI  
549 resolution. The value of this attribute MUST be the exact numeric version value of the XRI  
550 Resolution specification to which its containing XRD element conforms. See section 2.1.1.

551 For more about versioning of the XRI resolution protocol, see section 12.2.

### 552 3.3.3 Priority Attribute

553 Certain XRD elements involved in the XRI resolution process (`xrd:Ref`, `xrd:Service`, and  
554 `xrd:URI`) may be present multiple times in an XRDS document to describe different references,  
555 redundant service endpoints, or for other reasons. In this case XRD authors may use the global  
556 priority attribute to prioritize selection of these element instances. Like the priority attribute of DNS  
557 records, it accepts a non-negative integer value.

558 Following are the normative processing rules that apply whenever there is more than one  
559 instance of the same type of element selected in an XRD (if there is only one instance selected,  
560 the priority attribute is ignored.)

- 561 1. The client SHOULD select the element instance with the lowest numeric value of the  
562 priority attribute. For example, an element with priority attribute value of "10" should be  
563 selected before an element with a priority attribute value of "11", and an element with  
564 priority attribute value of "11" should be selected before an element with a priority  
565 attribute value of "15". Zero is the highest priority attribute value. Null is the lowest priority  
566 attribute value.
- 567 2. If an element has no priority attribute, its priority attribute value is considered to be null.
- 568 3. If two or more instances of the same element type have identical priority attribute values  
569 (including the null value), the client SHOULD select one of the instances at random. This  
570 client SHOULD NOT simply choose the first instance that appears in XML document  
571 order (this is important in order to support intentional load balancing).
- 572 4. An element selected according to these rules is referred to in this specification as "the  
573 highest priority element". If this element is subsequently disqualified from the set of  
574 qualified elements, the next element selected according to these rules is referred to as  
575 "the next highest priority element". If an XRI resolution operation specifying selection of



576 the highest priority element fails, the resolver SHOULD attempt to select the next highest  
577 priority element unless otherwise specified. This process SHOULD be continued for all  
578 other instances of the qualified elements until success is achieved or all instances are  
579 exhausted.

580 When setting priority attributes, it is recommended that XRI authorities follow the standard  
581 practice in DNS and set the default highest priority attribute value to "10".

### 582 **3.4 XRI Encoding Requirements**

583 The W3C XML 1.0 specification **[XML]** requires values of XML elements of type `xs:anyURI` to  
584 be valid IRIs. Thus all XRIs used as the values of XRD elements of this type MUST be in at least  
585 IRI-normal form as defined in section 2.3 of **[XRISyntax]**.

586 A further restriction applies to XRIs used in XRI resolution because it relies on HTTP(S) as a  
587 transport protocol. When an XRI is used as the value of an `xrd:Query`, `xrd:LocalID`,  
588 `xrd:XRD/xrd:Ref`, `xrd:Type`, or `xrd:Path` element, it MUST be in URI-normal form as  
589 defined in section 2.3 of **[XRISyntax]**.

590 Note that XRIs composed entirely of valid URI characters and that do not use XRI cross-  
591 reference syntax do not require escaping in the transformation to URI-normal form. However  
592 XRIs that use characters valid only in IRIs or that use XRI cross-reference syntax may require  
593 percent encoding in the transformation to URI-normal form as explained in section 2.3 of  
594 **[XRISyntax]**.

---

## 595 4 Inputs and Outputs

596 Logically, XRI resolution is a function invoked by an application to dereference an XRI into a  
597 media type that describes the target resource. This section defines the logical inputs and outputs  
598 of this function, however it does not specify any particular binding to any local resolver interface.  
599 For purposes of illustration, a non-normative, language-neutral API is suggested in Appendix C.  
600 A binding to an HTTP interface for XRI proxy resolvers is specified in section 7.

### 601 4.1 Inputs

602 Table 7 summarizes the logical input parameters to the authority phase of XRI resolution (section  
603 5). In this specification, references to these parameters will use the names in the first column.  
604 Local APIs MAY use different names for these parameters and MAY define additional  
605 parameters.

Logical Parameter Name	Type	Required/Optional	Default
QXRI (query XRI)	xs:anyURI	Required	N/A
Resolution Media Type	xs:string (media type)	Optional	Null

606 **Table 7: Input parameters for the authority phase of XRI resolution.**

607 Table 8 summarizes the additional input parameters used in the service endpoint selection phase  
608 of XRI resolution (section 7).

Logical Parameter Name	Type	Required/Optional	Default
Service Type	xs:anyURI	Optional	Null
Service Media Type	xs:string (media type)	Optional	Null

609 **Table 8: Input parameters for the service endpoint selection phase of XRI resolution.**

610 The following sections specify additional validation and usage requirements.

#### 611 4.1.1 QXRI (Authority String, Path String, and Query String)

612 The QXRI (query XRI) is the only REQUIRED input parameter. Per [XRISyntax], a QXRI consists  
613 of three logical subparameters as defined in Table 9.

Logical Parameter Name	Type	Required/Optional	Value
Authority String	xs:string	Required	Contents of the authority segment of the QXRI, not including leading double forward slashes ("/") or terminating single forward slash ("/").
Path String	xs:string	Optional	Contents of the path component of the QXRI, not including leading single forward slash ("/") or terminating delimiter (such as "/", "?", "#", white space, or CRLF). If the path component is absent or empty the value is null.
Query String	xs:string	Optional	Contents of the query component of the QXRI, not including leading question mark ("?") or terminating delimiter (such as "#", white space, or CRLF). If the query component is absent or empty the value is null.

614 **Table 9: Subparameters of the QXRI input parameter.**

615 The fourth possible component of a QXRI—a fragment—is by definition resolved locally relative  
616 to the target resource identified by the combination of the Authority, Path, and Query  
617 components, and as such does not play a role in XRI resolution.

618 Following are the constraints on the value of the QXRI parameter.

- 619 1. It MUST be a valid absolute XRI according to the ABNF defined in [XRISyntax]. To  
620 resolve a relative XRI reference, it must be converted into an absolute XRI using the  
621 procedure defined in section 2.4 of [XRISyntax].
- 622 2. For authority or proxy resolution as defined in this specification, the QXRI MUST be in  
623 URI-normal form as defined in section 2.3.1 of [XRISyntax]. A local resolver API MAY  
624 support the input of other normal forms but SHOULD document the normal form(s) it  
625 supports and its normalization policies.

#### 626 4.1.2 Resolution Media Type

627 The Resolution Media Type is an OPTIONAL string that is used to specify:

- 628 • The media type for the resolution response.
- 629 • Whether generic or trusted resolution must be used by the resolver.
- 630 • Whether references should be followed during resolution.
- 631 • Whether final service endpoint selection should be performed.

632 Following are the normative requirements for the use of this parameter.

- 633 1. The value of Resolution Media Type MUST be one of the values specified in Table 5 and  
634 MAY include any of the media type parameters specified in Table 6.
- 635 2. If the value of the `trust` media type parameter is `none` or null, or if this parameter is  
636 absent, the resolver MAY use its choice of authority resolution protocol.

- 637 3. If the value of the `trust` media type parameter is `https`, the resolver MUST use the  
638 HTTPS trusted authority resolution protocol specified in section 6.1 (or return an error  
639 saying this is not supported).
- 640 4. If the value of the `trust` media type parameter is `saml`, the resolver MUST use the  
641 SAML trusted authority resolution protocol specified in section 6.2 (or return an error  
642 saying this is not supported).
- 643 5. If the value of the `trust` media type parameter is `https+saml`, the resolver MUST use  
644 the HTTPS+SAML trusted authority resolution protocol specified in section 6.3 (or return  
645 an error saying this is not supported).
- 646 6. If the value of the `refs` media type parameter is `true` or null, or if the parameter is  
647 absent, the resolver MUST perform reference processing as defined in section 9 if it is  
648 necessary to complete resolution (or return an error saying this is not supported).
- 649 7. If the value of the `refs` media type parameter is `false`, the resolver MUST NOT  
650 perform reference processing during resolution.
- 651 8. If the value of the `sep` media type parameter is `true`, the resolver MUST perform final  
652 service endpoint selection (or return an error saying this is not supported).
- 653 9. If the value of the `sep` media type parameter is `false`, the resolver MUST NOT perform  
654 service endpoint selection.
- 655 10. If the `sep` media type parameter is absent or null, the resolver MUST NOT perform  
656 service endpoint selection if the value of Resolution Media Type is either  
657 `application/xrds+xml` or `application/xrd+xml`, but MUST perform service  
658 endpoint selection if the value of Resolution Media Type is any other value, including null.

659 Future versions of this specification, or other specifications for XRI resolution, MAY use other  
660 values for Resolution Media Type or its media type parameters.

### 661 4.1.3 Service Type

662 The Service Type is an OPTIONAL value of type `xs:anyURI` used to request a specific type of  
663 service in the service endpoint selection phase (section 7). The value of this parameter MUST be  
664 a valid absolute XRI, IRI, or URI in URI-normal form as defined by **[XRISyntax]**. (Note that URI-  
665 normal form is specified so that this parameter may be passed to a proxy resolver in a QXRI  
666 query parameter as defined in section 7.) The Service Type values defined for XRI resolution  
667 services are specified in section 2.1.2.

### 668 4.1.4 Service Media Type

669 The Service Media Type is an OPTIONAL string used to request a specific media type in the  
670 service endpoint selection phase (section 7). The value of this parameter MUST be a valid media  
671 type as defined by **[RFC2046]**. The Service Media Type values defined for XRI resolution  
672 services are specified in section 2.3.

## 673 4.2 Outputs

674 Table 10 summarizes the logical outputs of XRI resolution.

Logical Output Name	Media Type Value for Requesting Authority Resolution Only	Media Type Value for Requesting Final Service Endpoint Selection
XRDS Document	application/xrds+xml	application/xrds+xml;sep=true
XRD Document	application/xrd+xml	application/xrd+xml;sep=true
URI List	N/A	text/uri-list
HTTP(S) Redirect	N/A	<i>null</i>

675 **Table 10: Outputs of XRI resolution.**

676 The following sections provide additional construction and validation requirements.

### 677 4.2.1 XRDS Document

678 If the value of the Resolution Media Type parameter is `application/xrds+xml`, the following  
679 rules apply.

- 680 1. The output **MUST** be a valid XRDS document according to the schema defined in  
681 Appendix A. Also, any nested XRDS documents included as a result of reference  
682 processing must also be valid.
- 683 2. Each of the contained XRD elements must be a valid XRD document according to the  
684 schema defined in Appendix A.
- 685 3. The XRD elements **MUST** conform to the additional requirements in section 3.
- 686 4. If the value of the `trust` media type parameter is `saml` or `https+saml`, the XRD  
687 element **MUST** further conform to the additional requirements in section 6.2.
- 688 5. If the value of the `sep` media type parameter is `true`, service endpoint selection **MUST**  
689 be performed as defined in section 8, even if the values of all three service endpoint  
690 selection input parameters (Service Type, Service Media Type, and Path String) are null.
- 691 6. If reference processing is necessary during the service endpoint selection process, the  
692 final child element of the root level `xrds:XRDS` element **MUST** be a nested XRDS  
693 document as specified in section 9. Any other filtering of the final XRD element **MUST**  
694 NOT be performed.
- 695 7. If the output is an error, this error **MUST** be returned using the `xrd:Status` element of  
696 the final XRD in the XRDS document as defined in section 10.

### 697 4.2.2 XRD Document

698 If the value of the Resolution Media Type parameter is `application/xrd+xml`, the following  
699 rules apply.

- 700 1. The output **MUST** be a valid XRD document according to the schema defined in  
701 Appendix A.
- 702 2. The XRD elements **MUST** conform to the additional requirements in section 3.
- 703 3. If the value of the `sep` media type parameter is `false` or null, or if this parameter is  
704 absent, the XRD **MUST** be the final XRD in the XRDS document produced as a result of

705 authority resolution. Service endpoint selection or any other filtering of the XRD  
706 document MUST NOT be performed.

- 707 4. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST  
708 be performed as defined in section 8, even if the values of all three service endpoint  
709 selection input parameters (Service Type, Service Media Type, and Path String) are null.
- 710 5. If service endpoint selection is performed, the XRD document MUST only include the  
711 `xrd:Service` elements selected according to the rules specified in section 8 (or return  
712 an error if no `xrd:Service` elements were selected). In addition, all XRD elements that  
713 are subject to the global `xrd:priority` attribute (even if the attribute is absent or null)  
714 MUST be returned in order of highest to lowest priority as defined in section 3.3.3. Any  
715 other filtering of the XRD document MUST NOT be performed.
- 716 6. If the output is an error, this error MUST be returned using the `xrd:Status` element as  
717 defined in section 10.

### 718 4.2.3 URI List

719 If the value of the Resolution Media Type parameter is `text/uri-list`, the following rules  
720 apply.

- 721 1. For this output, service endpoint selection is REQUIRED.
- 722 2. If authority resolution and service endpoint selection are both successful, the output  
723 MUST be a valid URI List as defined by section 5 of [RFC2483].
- 724 3. If, after applying the service endpoint selection rules, more than one service endpoint is  
725 selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as  
726 defined in section 3.3.3.
- 727 4. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)  
728 MUST be constructed as defined in section 8.4.
- 729 5. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`  
730 elements within the selected `xrd:Service` element as defined in section 3.3.3. When  
731 two or more of the source `xrd:URI` elements have equal priority, their constructed URIs  
732 SHOULD be returned in random order. Any other filtering of the URI list MUST NOT be  
733 performed.
- 734 6. If the output is an error, it MUST be returned with the content type `text/plain` as  
735 defined in section 10.

### 736 4.2.4 HTTP(S) Redirect

737 In XRI proxy resolution, the Resolution Media Type parameter may be null. In this case the output  
738 of a proxy resolver is an HTTP(S) redirect as defined in section 7.6.

---

## 739 5 Generic Authority Resolution

740 Authority resolution is the first phase of XRI resolution as described in section 1.1. It applies only  
741 to the Authority String of the QXRI. This may be either an *XRI authority* or an *IRI authority* as  
742 described in section 2.2.1 of [XRISyntax].

743 XRI authorities and IRI authorities have different syntactic structures, partially due to the higher  
744 level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved  
745 to XRDS documents one subsegment at a time as specified in section 5.1. IRI authorities, since  
746 they are based on DNS names or IP addresses, are resolved into an XRDS document through a  
747 special HTTP(S) request using the entire IRI authority segment as specified in section 5.2.

### 748 5.1 XRI Authority Resolution

#### 749 5.1.1 Service Type and Service Media Type

750 The protocol defined in this section is identified by the values in Table 11.

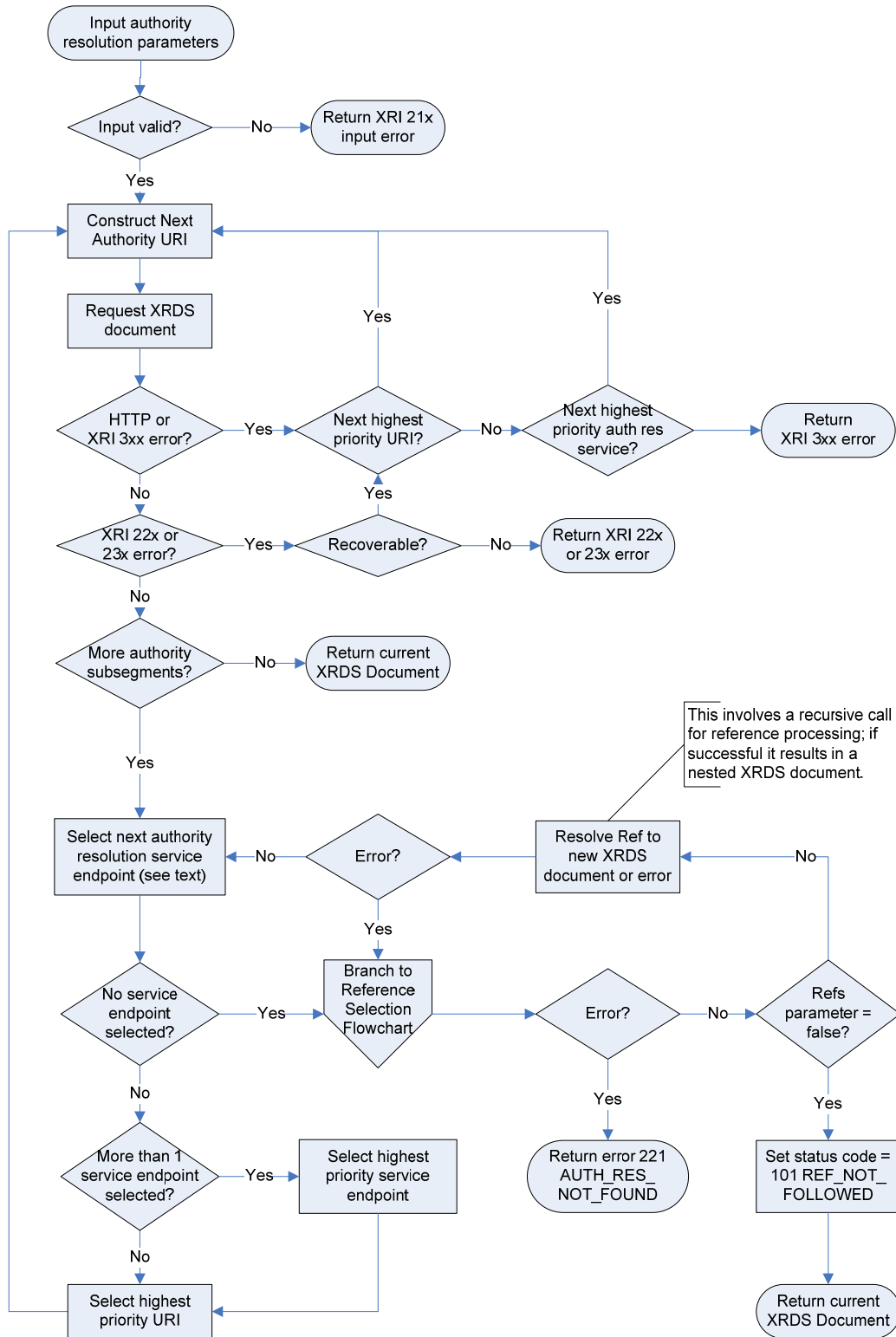
Service Type	Service Media Type	Media Type Parameters
xri://\$res*auth*(\$v*2.0)	application/xrds+xml	trust=none (this is the default – see below)

751 **Table 11: Service Type and Service Media Type values for generic authority resolution.**

752 A generic authority resolution service endpoint advertised in an XRDS document MUST use the  
753 Service Type identifier and Service Media Type identifier defined in Table 11. Because  
754 `trust=none` is the default value of trust parameter, the values `application/xrds+xml` and  
755 `application/xrds+xml;trust=none` MUST be considered equivalent for the purposes of  
756 both service endpoint selection and HTTP(S) Accept header values as described below.

757 **5.1.2 Protocol**

758 Figure 2 shows the overall logical flow of the generic authority resolution process.



759  
760

**Figure 2: Authority resolution flowchart**



761 Following are the normative requirements for behavior of an XRI resolver and an XRI resolution  
762 service for performing generic XRI authority resolution:

- 763 1. The resolver MUST be preconfigured with (or have a means of obtaining) the XRD  
764 document describing the community root authority for the XRI to be resolved as defined  
765 in section 5.1.3.
- 766 2. Resolution of each subsegment in the Authority String after the community root  
767 subsegment MUST proceed in subsegment order (left-to-right) using fully qualified  
768 subsegment values as defined in section 5.1.4. If the final output is an XRDS document,  
769 this document MUST contain an ordered list of `xrd:XRDS` elements—one for each  
770 authority subsegment successfully resolved by the resolver client. This list MUST appear  
771 in the same order as the corresponding subsegments in the Authority String. In addition,  
772 any references followed MUST be represented by nested `xrd:XRDS` documents  
773 immediately following the `xrd:XRDS` element containing the reference as defined in  
774 section 9.
- 775 3. Subsegments that use XRI cross-reference syntax MUST be resolved as defined in  
776 section 5.1.5.
- 777 4. The resolver MAY request that a recursing authority resolution service perform resolution  
778 of multiple subsegments as defined in section 5.1.6.
- 779 5. For each iteration of the authority resolution process, the next authority resolution service  
780 endpoint MUST be selected as defined in section 7 using the values from Table 11. In  
781 addition, the Next Authority URI MUST be constructed as defined in section 5.1.7.
- 782 6. Each authority subsegment MUST be resolved via an HTTP(S) GET request to the Next  
783 Authority URI.
- 784 7. The HTTP(S) request MUST contain an Accept header with the media type identifier  
785 defined in Table 11. This media type identifier MUST be interpreted as the value of the  
786 Resolution Media Type input parameter.
- 787 8. The ultimate HTTP(S) response from an authority resolution service to a successful  
788 resolution request MUST contain either: a) a 2XX response with a valid XRDS document  
789 containing an XRD element for each authority subsegment resolved, or b) a 304  
790 response signifying that the cached version on the resolver is still valid (depending on the  
791 client's HTTP(S) request). There is no restriction on intermediate redirects (i.e., 3XX  
792 result codes) or other result codes (e.g., a 100 HTTP response) that eventually result in a  
793 2XX or 304 response through normal operation of **[RFC2616]**.
- 794 9. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in  
795 the resolution process and the resolver SHOULD return the appropriate error code and  
796 context message as specified in section 10. In recursive resolution, such an error MUST  
797 be returned by the recursing authority resolution service to the resolver as specified in  
798 section 10.3.
- 799 10. If an XRD does not include the next requested authority service endpoint but includes  
800 one or more `xrd:Ref` elements, the resolver MUST perform reference processing as  
801 defined in section 9 unless the `refs` media type parameter defined in Table 6 is set to  
802 `false`. If the `refs` media type parameter is set to `false` and the XRD contains at least  
803 one `xrd:Ref` element that could be followed, the resolver MUST return a successful  
804 response with a status code of 101 `REF_NOT_FOLLOWED`. (Note that such reference  
805 processing, if successful, will result in a separate nested XRDS document describing the  
806 resolved reference.)
- 807 11. A successful response that does not include the next required authority service endpoint  
808 in the XRD and does not include any `xrd:Ref` elements MUST return an error with a  
809 status code of 221 `AUTH_RES_NOT_FOUND` regardless of the value of the `refs` media  
810 type parameter.

811 12. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section  
812 11. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent  
813 possible to maintain the efficiency and scalability of the HTTP-based resolution system.  
814 The recommended use of HTTP caching headers is described in more detail in section  
815 11.2.1.

### 816 5.1.3 Community Root Authorities

817 Identifier management policies are defined on a community-by-community basis. For XRI  
818 authorities, the resolution community is specified by the first (leftmost) subsegment of the  
819 authority segment of the XRI. This is referred to as the *community root authority*. When a  
820 resolution community chooses to create a new community root authority, it SHOULD define  
821 policies for assigning and managing identifiers under this authority. Furthermore, it SHOULD  
822 define what resolution protocol(s) may be used for these identifiers.

823 For an XRI authority, the community root may be either a global context symbol (GCS) character  
824 or top-level cross-reference as specified in section 2.2.1.1 of **[XRISyntax]**. In either case, the  
825 corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution  
826 service endpoints for that community.

827 This community root XRDS document, or its location, must be known *a priori* and is part of the  
828 configuration of an XRI resolver, similar to the specification of root DNS servers for a DNS  
829 resolver. Note that is not strictly necessary to publish this information in an XRDS document—it  
830 may be supplied in any format that enables configuration of the XRI resolvers in the community.  
831 However publishing an XRDS document at a known location simplifies this process. It is also a  
832 recommended best practice for this XRDS document to contain:

- 833 • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- 834 • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML  
835 trusted resolution is supported.
- 836 • Both of the above if HTTPS+SAML trusted resolution is supported.
- 837 • The service endpoints and supported media types of the community's XRI proxy  
838 resolver(s) if proxy resolution is supported.

839 For a list of public community root authorities and the locations of their community root XRDS  
840 documents, see the XRI Technical Committee home page at [http://www.oasis-  
841 open.org/committees/xri](http://www.oasis-open.org/committees/xri).

### 842 5.1.4 Qualified Subsegments

843 A qualified subsegment is defined by the productions whose names start with “xri-subseg” in  
844 section 2.2.3 of **[XRISyntax]** including the leading syntactic delimiter (“\*” or “!”). A qualified  
845 subsegment MUST include the leading syntactic delimiter even if it was optionally omitted in the  
846 original XRI (see section 2.2.3 of **[XRISyntax]**).

847 If the first subsegment of an XRI authority is a GCS character and the following subsegment does  
848 not begin with a “\*” (indicating a reassignable subsegment) or a “!” (indicating a persistent  
849 subsegment), then a “\*” is implied and MUST be added when constructing the qualified  
850 subsegment as specified in section 5.1.7. Table 12 and Table 13 illustrate the differences  
851 between parsing a reassignable subsegment following a GCS character and parsing a cross-  
852 reference, respectively.

853

<b>XRI</b>	xri://@example*internal/foo
<b>XRI Authority</b>	@example*internal
<b>Community Root Authority</b>	@
<b>First Qualified Subsegment Resolved</b>	*example

854 **Table 12: Parsing the first subsegment of an XRI that begins with a global context symbol.**

<b>XRI</b>	xri://(http://www.example.com)*internal/foo
<b>XRI Authority</b>	(http://www.example.com)*internal
<b>Community Root Authority</b>	(http://www.example.com)
<b>First Qualified Subsegment Resolved</b>	*internal

855 **Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.**

### 856 5.1.5 Cross-References

857 Any subsegment within an XRI authority segment may be a cross-reference (see section 2.2.2 of  
 858 **[XRISyntax]**.) Cross-references are resolved identically to any other subsegment because the  
 859 cross-reference is considered opaque, i.e., the value of the cross-reference (including the  
 860 parentheses) is the literal value of the subsegment for the purpose of resolution.

861 Table 14 provides several examples of resolving cross-references. In these examples,  
 862 subsegment "b" resolves to a Next Authority Service Endpoint URI of "http://example.com/xri-  
 863 authority/" and recursing authority resolution is not being requested.

864

<b>Cross-reference type</b>	<b>Example XRI</b>	<b>Next Authority URI after resolving "xri://@!a!b"</b>
Absolute XRI	xri://@!a!b!(@!1!2!3)*e/f	http://example.com/xri-authority/!(@!1!2!3)
Absolute URI	xri://@!a!b*(mailto:jd@example.com)*e/f	http://example.com/xri-authority/*(mailto:jd@example.com)
Absolute XRI w/ XRI metadata	xri://@!a!b*(\$v/2.0)*e/f	http://example.com/xri-authority/*(\$v*2.0)
Relative XRI	xri://@!a!b*(c*d)*e/f	http://example.com/xri-authority/*(c*d)
Relative URI	xri://@!a!b*(foo/bar)*e/f	http://example.com/xri-authority/*(foo%2fbar)

865 **Table 14: Examples of the Next Authority URIs constructed using different types of cross-references.**

### 866 5.1.6 Recursing Authority Resolution

867 If an authority resolution service offers recursing resolution, an XRI resolver may request  
 868 resolution of multiple authority subsegments in one transaction. If a resolver makes such a  
 869 request, the responding authority resolution service MAY perform the additional recursing  
 870 resolution steps requested. In this case the recursing authority resolution service acts as a  
 871 resolver to the other authority resolution service endpoints that need to be queried. Alternatively,  
 872 the recursing authority resolution service may retrieve XRDs from its local cache until it reaches a

873 subsegment whose XRD is not locally cached, or it may simply recurse only as far as it is  
874 authoritative. If an authority resolution service performs any recursing resolution, it MUST return  
875 an ordered list of `xrd:XRDS` elements (and nested `xrd:XRDS` elements if references are  
876 followed) in an `xrd:XRDS` document for all subsegments resolved as defined in section 5.1.2.  
877 The recursing authority resolution service MAY resolve fewer subsegments than requested by the  
878 resolver. The recursing authority resolution service is under no obligation to resolve more than  
879 the first subsegment (for which it is, by definition, authoritative).  
880 If the recursing authority resolution service does not resolve the entire set of subsegments  
881 requested, the resolver is responsible for continuing the authority resolution process itself. At any  
882 stage, however, the resolver MAY request that the next authority resolution service recursively  
883 resolve any remaining subsegments.

## 884 5.1.7 Construction of the Next Authority URI

885 At each step in authority resolution, a URI must be constructed for the next HTTP(S) request.  
886 This URI is constructed by the XRI resolver from two strings—one representing the next authority  
887 resolution service endpoint selected from the current XRD, and the other representing the next  
888 unresolved subsegment or group of subsegments in the QXRI Authority String.

889 The process for selecting the next authority resolution service endpoint from the current XRD is  
890 defined in section 8. For generic authority resolution, the process MAY select the this service  
891 endpoint from the parameters defined in Table 11, or from any of the parameters defined for  
892 selection of trusted resolution service endpoints in section 6.

893 From the output of the service endpoint selection process, the resolver MUST select the highest  
894 priority URI of the highest priority authority resolution service endpoint. This is called the *Next  
895 Authority Service Endpoint URI*. If this URI does not end with a forward slash (“/”), one MUST be  
896 appended before proceeding.

897 The second string is called the *Next Authority String* and it consists of either:

- 898 • The next fully qualified subsegment to be resolved (see section 5.1.4), or
- 899 • In the case of recursing resolution, the next fully qualified subsegment plus any additional  
900 subsegments for which recursing resolution is requested (see section 5.1.5).

901 The final step is to append the Next Authority String to the path component of the Next Authority  
902 Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

903 Construction of the Next Authority URI is more formally described in this pseudo-code for  
904 resolving a “next-auth-string” via a “next-auth-sep-uri”:

```
905 if (path portion of next-auth-sep-uri does not end in "/"):  
906     append "/" to path portion of next-auth-sep-uri  
907  
908 if (next-auth-string is not preceded with "*" or "!" delimiter):  
909     prepend "*" to next-auth-string  
910  
911 append uri-escape(next-auth-string) to path of next-auth-sep-uri
```

## 912 5.2 IRI Authority Resolution

913 From the standpoint of generic authority resolution, an IRI authority segment represents either a  
914 DNS name or an IP address at which an XRDS document describing the authority may be  
915 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET  
916 request to a URI constructed from the IRI authority segment. The resulting XRDS document can  
917 then be consumed in the same manner as one obtained using XRI authority resolution.

918 While the use of IRI authorities provides backwards compatibility with the large installed base of  
919 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of

920 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities  
921 are not recommended for new deployments of XRI identifiers.

922 This section defines IRI authority resolution as a simple extension to the XRI authority resolution  
923 protocol defined in the preceding section.

## 924 5.2.1 Service Type and Media Type

925 Because IRI authority resolution takes place at a level “below” XRI authority resolution, it cannot  
926 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority  
927 resolution uses the same media type as generic XRI authority resolution.

## 928 5.2.2 Protocol

929 Following are the normative requirements for IRI authority resolution that differ from generic XRI  
930 authority resolution:

- 931 1. The next authority URI is constructed by extracting the entire IRI authority segment and  
932 prepending the string “http://”. See the exception in section 5.2.3.
- 933 2. The HTTP GET request MUST include an HTTP Accept header containing only the  
934 following:

935 `Accept: application/xrds+xml`

- 936 3. The HTTP GET request MUST have a Host: header (as defined in section 14.23 of  
937 **[RFC2616]**) containing the value of the IRI authority segment.
- 938 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document  
939 containing the XRD describing that authority.
- 940 5. The responding server MUST use the value of the Host header to populate the  
941 `xrd:XRD/xrd:Query` element in the resulting XRD. For example:

942 `Host: example.com`

943 Note that because IRI authority resolution is required to process the entire IRI authority segment  
944 in a single step, recursing authority resolution does not apply.

## 945 5.2.3 Optional Use of HTTPS

946 Section 6 of this specification defines trusted resolution only for XRI authorities. Trusted  
947 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to  
948 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY  
949 use HTTPS as the access protocol for retrieving the authority's XRD. If the resolver is satisfied,  
950 via transport level security mechanisms, that the response is from the expected IRI authority, the  
951 resolver may consider this an HTTPS trusted resolution response as defined in section 6.1.

---

## 6 Trusted Authority Resolution

952

953 This section defines three options for performing trusted XRI authority resolution as an extension  
954 of the generic XRI authority resolution service defined in section 5.1—one using HTTPS, one  
955 using SAML assertions, and one using both.

### 6.1 HTTPS

956

957 This option for trusted authority resolution is a very simple addition to generic authority resolution  
958 in which all communication with authority resolution service endpoints is carried out over HTTPS.  
959 This provides transport-level security and server authentication, however it does not provide  
960 message-level security or a means for a responder to provide different responses for different  
961 requestors.

#### 6.1.1 Service Type and Service Media Type

962

963 The protocol defined in this section is identified by the values in Table 15.

Service Type	Service Media Type	Media Type Parameters
xri://\$res*auth*(\$v*2.0)	application/xrds+xml	trust=https

964

**Table 15: Service Type and Service Media Type values for HTTPS trusted authority resolution.**

965 An HTTPS trusted resolution service endpoint advertised in an XRDS document **MUST** use the  
966 Service Type identifier and Service Media Type identifier (including the `trust=https`  
967 parameter) defined in Table 15. In addition, the identifier authority **MUST** use an HTTPS URI as  
968 the value of the `xrd:URI` element(s) for this service endpoint.

#### 6.1.2 Protocol

969

970 Following are the normative requirements for HTTPS trusted authority resolution that differ from  
971 generic XRI authority resolution (section 5.1):

- 972 1. All authority resolution service endpoints **MUST** be selected using the values defined in  
973 Table 15.
- 974 2. All authority resolution requests including the starting request to a community root  
975 authority **MUST** use the HTTPS protocol as defined in [RFC2818]. A successful HTTPS  
976 response **MUST** be received from each authority in the resolution chain or the resolver  
977 **MUST** output an error.
- 978 3. All authority resolution requests **MUST** contain an HTTPS Accept header with the media  
979 type identifier (including the `trust=https` parameter) defined in Table 15. This **MUST**  
980 be interpreted as the value of the Resolution Media Type input parameter.
- 981 4. If the resolver finds that an authority in the resolution chain does not support HTTPS, the  
982 resolver **MUST** return a 23x error as defined in section 10.

## 6.2 SAML

983

984 In SAML trusted resolution, the resolver requests a content type of  
985 `application/xrds+xml;trust=saml` and the authority resolution service responds with an  
986 XRDS document containing an XRD with an additional element—a digitally signed SAML [SAML]  
987 assertion that asserts the validity of the containing XRD. SAML trusted resolution provides  
988 message integrity but does not provide confidentiality. The latter may be achieved by combining it

989 with HTTPS as defined in section 6.3. Message confidentiality may also be achieved with other  
990 security protocols used in conjunction with this specification. SAML trusted resolution also does  
991 not provide a means for an authority to provide different responses for different requestors; client  
992 authentication is explicitly out-of-scope for version 2.0 of XRI resolution.

## 993 6.2.1 Service Type and Service Media Type

994 The protocol defined in this section is identified by the values in Table 16.

Service Type	Service Media Type	Media Type Parameters
xri://\$res*auth*(\$v*2.0)	application/xrds+xml	trust=saml

995 **Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.**

996 A SAML trusted resolution service endpoint advertised in an XRD document MUST use the  
997 Service Type identifier and Service Media Type identifier (including the `trust=saml` parameter)  
998 defined in Table 16. In addition, for transport security the identifier authority SHOULD offer at  
999 least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

## 1000 6.2.2 Protocol

### 1001 6.2.2.1 Client Requirements

1002 For a resolver, trusted resolution is identical to the generic resolution protocol (section 5.1) with  
1003 the addition of the following requirements:

- 1004 1. All authority resolution service endpoints MUST be selected using the values defined in  
1005 Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an  
1006 authority unless the authority advertises a resolution service endpoint matching these  
1007 values.
- 1008 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is  
1009 recommended for confidentiality.
- 1010 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the  
1011 media type identifier (including the `trust=saml` parameter) defined in Table 16. This  
1012 MUST be interpreted as the value of the Resolution Media Type input parameter. (Clients  
1013 willing to accept either generic or trusted responses may use a combination of media  
1014 type identifiers in the Accept header as described in section 14.1 of [RFC2616]. Media  
1015 type identifiers SHOULD be ordered according to the client's preference for the media  
1016 type of the response. If a client performing generic authority resolution receives an XRD  
1017 containing SAML elements, it is NOT REQUIRED to validate the signature or perform any  
1018 processing of these elements.)
- 1019 4. A resolver MAY request recursing authority resolution of multiple subsegments as  
1020 defined in section 6.2.3.
- 1021 5. The resolver MUST individually validate each XRD in the resolution chain according to  
1022 the rules defined in section 6.2.4. When `xrd:XRD` elements come both from freshly-  
1023 retrieved XRD documents and from a local cache, a resolver MUST ensure that these  
1024 requirements are satisfied each time a resolution request is performed.

### 1025 6.2.2.2 Server Requirements

1026 For an authority resolution service, trusted resolution is identical to the generic resolution protocol  
1027 (section 5.1) with the addition of the following requirements:

- 1028 1. The HTTP(S) response to a trusted resolution request MUST include a content type of  
1029 "application/xrds+xml;trust=saml".

- 1030 2. The XRDS document returned by the resolution service MUST contain a  
1031 `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid  
1032 per the processing rules described by **[SAML]**.
- 1033 3. The SAML assertion MUST contain a valid enveloped digital signature as defined by  
1034 **[XMLDSig]** and as constrained by section 5.4 of **[SAML]**.
- 1035 4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML  
1036 assertion. Specifically, the signature MUST contain a single  
1037 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference  
1038 MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML  
1039 assertion. The `URI` reference MUST NOT be empty and it MUST refer to the identifier  
1040 contained in the `xrd:XRD/@xml:id` attribute.
- 1041 5. In **[SAML]**, the digital signature enveloped by the SAML assertion may contain a  
1042 `ds:KeyInfo` element. If this element is included, it MUST describe the key used to verify  
1043 the digital signature element. Because the signing key is known in advance by the  
1044 resolution client, the `ds:KeyInfo` element SHOULD be omitted from the digital  
1045 signature.
- 1046 6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST  
1047 match the XRI authority subsegment requested by the client.
- 1048 7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match  
1049 the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD  
1050 advertising availability of trusted resolution service from this authority as required in  
1051 section 6.2.5.
- 1052 8. The `xrd:XRD/saml:Subject/saml:NameID` element MUST be present and equal to  
1053 the `xrd:XRD/xrd:Query` element.
- 1054 9. The `NameQualifier` attribute of the  
1055 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be  
1056 present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.
- 1057 10. There MUST be exactly one `saml:AttributeStatement` present in the  
1058 `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute`  
1059 element with a `Name` attribute of `"xri://$xrd*($v*2.0)"`. This `saml:Attribute`  
1060 element MUST contain exactly one `saml:AttributeValue` element whose text value  
1061 is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that is the immediate  
1062 parent of the `saml:Assertion` element.

### 1063 **6.2.3 Recursing Authority Resolution**

1064 If a resolver requests trusted resolution of multiple authority subsegments (see section 5.1.5), a  
1065 recursing authority resolution service SHOULD attempt to perform trusted resolution on behalf of  
1066 the resolver as described in this section. However if the resolution service is not able to obtain  
1067 trusted XRDs for one or more additional recursing subsegments, it SHOULD return only the  
1068 trusted XRDs it has obtained and allow the resolver to continue.

### 1069 **6.2.4 Client Validation of XRDs**

1070 For each XRD returned as part of a trusted resolution request, the resolver MUST validate the  
1071 XRD according to the rules defined in this section.

- 1072 1. The `xrd:XRD/saml:Assertion` element MUST be present.
- 1073 2. This assertion MUST valid per the processing rules described by **[SAML]**.
- 1074 3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by  
1075 **[XMLDSig]** and constrained by Section 5.4 of **[SAML]**.



- 1076 4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML  
1077 assertion. Specifically, the signature MUST contain a single  
1078 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference  
1079 MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent  
1080 of the signed SAML assertion.
- 1081 5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo`  
1082 element, the resolver MAY reject the signature if this key does not match the signer's  
1083 expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor  
1084 that was used to describe the current authority. See section 6.2.5.
- 1085 6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose  
1086 resolution resulted in the current XRD.
- 1087 7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the  
1088 `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability  
1089 of trusted resolution service from this authority as required in section 6.2.5.
- 1090 8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the  
1091 `NameQualifier` attribute of the  
1092 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1093 9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the  
1094 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1095 10. There MUST exist exactly one  
1096 `xrd:XRD/saml:Assertion/saml:AttributeStatement` with exactly one  
1097 `saml:Attribute` element that has a `Name` attribute of "`xri://$xrd*($v*2.0)`". This  
1098 `saml:Attribute` element must have exactly one `saml:AttributeValue` element  
1099 whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that  
1100 is the immediate parent of the signed SAML assertion.

1101 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result  
1102 MUST NOT be considered a valid trusted resolution response as defined by this specification.  
1103 Note that this does not preclude a resolver from considering alternative resolution paths. For  
1104 example, if an XRD advertising SAML trusted resolution service has two or more  
1105 `xrd:XRD/xrd:Service/xrd:URI` elements and the response from one service endpoint fails  
1106 to meet the requirements above, the client MAY repeat the validation process using the second  
1107 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as  
1108 defined by this document and SAML trusted resolution may continue.

## 1109 6.2.5 Correlation of ProviderID and KeyInfo Elements

1110 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at  
1111 least one unique persistent service provider identifier expressed in the  
1112 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority  
1113 resolution service. This `ProviderID` value MUST NOT ever be reassigned to another XRI  
1114 authority. A `ProviderID` may be any valid URI that meets these requirements of persistence and  
1115 uniqueness. Examples of appropriate URIs include URNs as defined by [RFC2141] and fully  
1116 persistent XRIs converted to URI-normal form as defined by [XRISyntax].

1117 The purpose of `ProviderIDs` in XRI resolution is to enable resolvers to correlate the metadata in  
1118 an XRD advertising trusted authority resolution service with the response received from a trusted  
1119 resolution service endpoint. If the signed XRD response contains the same `ProviderID` as the  
1120 XRD used to advertise a service, and the resolver has reason to trust the signature, the resolver  
1121 can trust that the XRD response has not been maliciously replaced with another XRD.

1122 There is no defined discovery process for the `ProviderID` for a community root authority; it must  
1123 be published in the community root XRDS document (or other equivalent description—see  
1124 section 5.1.3) and verified independently. Once the community root XRDS document is known,

1125 the ProviderID for delegated XRI authorities within this community MAY be discovered using the  
1126 `xrd:XRD/xrd:Service/xrd:ProviderID` element of authority resolution service endpoints.  
1127 This trust mechanism may also be used for other services offered by an authority.

1128 In addition, the metadata necessary for SAML trusted authority resolution or other SAML [SAML]  
1129 interactions MAY be discovered using the `ds:KeyInfo` element (section 3.2.) Again, if this  
1130 element is present in an XRD advertising SAML authority resolution service (or any other  
1131 service), and the client has reason to trust this XRD, the client MAY use the associated  
1132 ProviderID to correlate the contents of this element with a signed response.

1133 To assist resolvers in using this key discovery mechanism, it is important that trusted authority  
1134 resolution services be configured to sign responses in such a way that the signature can be  
1135 verified using the correlated `ds:KeyInfo` element. For more information, see [SAML].

## 1136 6.3 HTTPS+SAML

### 1137 6.3.1 Service Type and Service Media Type

1138 The protocol defined in this section is identified by the values in Table 17.

Service Type	Service Media Type	Media Type Parameters
<code>xri://\$res*auth*(\$v*2.0)</code>	<code>application/xrds+xml</code>	<code>trust=https+saml</code>

1139 **Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.**

1140 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST  
1141 use the Service Type identifier and Service Media Type identifier (including the  
1142 `trust=https+saml` parameter) defined in Table 17. In addition, the identifier authority MUST  
1143 use an HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

### 1144 6.3.2 Protocol

1145 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- 1146 1. All authority resolution service endpoints MUST be selected using the values defined in  
1147 Table 17.
- 1148 2. All authority resolution requests and responses, including the starting request to a  
1149 community root authority, MUST conform to both the requirements of the HTTPS trusted  
1150 resolution protocol defined in section 6.1 and the SAML trusted resolution protocol  
1151 defined in section 6.2.
- 1152 3. All authority resolution requests MUST contain an HTTPS Accept header with the media  
1153 type identifier (including the `trust=https+saml` parameter) defined in Table 17. This  
1154 MUST be interpreted as the value of the Resolution Media Type input parameter.
- 1155 4. If the resolver finds that an authority in the resolution chain does not support both HTTPS  
1156 and SAML, the resolver MUST return a 23x error as defined in section 10.

---

## 7 Proxy Resolution

1157

1158 The preceding sections have defined XRI resolution as a set of logical functions that may  
1159 implemented via a local resolver interface. This section defines a mapping of these functions to  
1160 an HTTP(S) interface for remote invocation. This mapping includes a standard syntax for  
1161 expressing an XRI as an HTTP URI, called an *HXRI*, and a method of passing the other XRI  
1162 resolution input parameters as query parameters in the HXRI.

1163 Proxy resolution is useful for many reasons:

- 1164 • Offloading XRI resolution and service endpoint selection processing from a client to an  
1165 HTTP(S) server.
- 1166 • Optimizing XRD caching for a resolution community (a *caching proxy resolver*). Proxy  
1167 resolvers SHOULD use caching to resolve the same QXRI or QXRI components for  
1168 multiple clients as defined in section 11.4.
- 1169 • Returning HTTP(S) redirects to clients such as browsers that have no native  
1170 understanding of XRIs but can process HXRIs. This provides backwards compatibility  
1171 with the large installed base of existing HTTP clients.

### 7.1 Service Type and Media Types

1172

1173 The protocol defined in this section is identified by the values in Table 18.

Service Type	Service Media Types	Media Type Parameters
xri://\$res*proxy*(\$v*2.0)	application/xrds+xml application/xrd+xml text/uri-list	trust=none trust=https trust=saml trust=https+saml refs=true refs=false sep=true sep=false

1174

**Table 18: Service Type and Service Media Type values for proxy resolution.**

1175 A proxy resolution service endpoint advertised in an XRDS document MUST use the Service  
1176 Type identifier and Service Media Type identifiers (including the optional parameters) defined in  
1177 Table 18. In addition, if the media type parameters `trust=https` or `trust=https+saml` is  
1178 included, the identifier authority MUST offer at least one HTTPS URI as the value of the `xrd:URI`  
1179 element(s) for this service endpoint.

1180 It may appear to be of limited value to advertise proxy resolution service in an XRDS document if  
1181 a resolver must already know how to perform local XRI resolution in order to retrieve this  
1182 document. However advertising a proxy resolution service in the XRDS document for a  
1183 community root authority (section 5.1.3) can be very useful for applications that need to consume  
1184 XRI proxy resolution services or automatically generate HXRIs for resolution by non-XRI-aware  
1185 clients in that community. Those applications may discover the current URI(s) and media type  
1186 capabilities of a proxy resolver from this source.

1187 **7.2 HXRIs**

1188 The first step in an HTTP binding of the XRI resolution interface is to specify how the QXRI  
1189 parameter is passed within an HTTP(S) URI. Besides providing a binding for proxy resolution,  
1190 defining a standard syntax for expressing an XRI as an HTTP XRI (HXRI) has two other benefits:

- 1191 • It allows XRIs to be used anyplace an HTTP URI can appear, including in Web pages,  
1192 electronic documents, email messages, instant messages, etc.
- 1193 • It allows XRI-aware processors and search agents to recognize an HXRI and extract the  
1194 embedded XRI for direct resolution, processing, and indexing.

1195 To make this syntax as simple as possible for XRI-aware processors or search agents to  
1196 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority segment that  
1197 begins with the domain name "xri.". The QXRI is then appended as the entire local path (and  
1198 query component, if present). In essence, the proxy resolver URI including the forward slash after  
1199 the domain name serves as a machine-readable prefix for an absolute XRI.

1200 Following is the normative ABNF for an HXRI is defined below based on the XRI and ireg-  
1201 name productions defined in [XRISyntax]. Authors of XRIs that need to be understood by non-  
1202 XRI-aware clients SHOULD publish them as HTTP URIs conforming to this HXRI production.

```
1203 HXRI           = proxy-resolver "/" QXRI
1204 proxy-resolver = ( "http://" / "https://" ) proxy-reg-name
1205 proxy-reg-name = "xri." ireg-name
1206 QXRI          = XRI
```

1207 URI processors that recognize XRIs SHOULD interpret the local part of any HTTP or HTTPS URI  
1208 that conforms to this ABNF as an XRI provided the domain name is at least three levels deep  
1209 (e.g., "xri.example.com".) If the URI conforms to this ABNF but the domain name is only two  
1210 levels deep, URI processors SHOULD interpret the local part as an XRI only if the second-level  
1211 domain is known to offer XRI proxy resolution services.

1212 For references to communities that offer public XRI proxy resolution services, see the XRI  
1213 Technical Committee home page at <http://www.oasis-open.org/committees/xri>.

1214 **7.3 QXRI Query Parameters**

1215 Besides the QXRI, there are three other logical input parameters to XRI authority resolution and  
1216 service endpoint selection as defined in section 4.1: Resolution Media Type, Service Type, and  
1217 Service Media Type. These parameters are bound to an HTTP(S) interface using the  
1218 conventional web model of passing parameters in the query component of the HTTP URI (which  
1219 in the case of an HXRI means the query component of the QXRI). The binding of the logical  
1220 parameter names to QXRI query parameter names is defined in Table 19.

Logical Parameter Name	QXRI Query Parameter Name
Resolution Media Type	_xrd_r
Service Type	_xrd_t
Service Media Type	_xrd_m

1221 **Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.**

1222 Following are the rules for the use of the parameters specified in Table 19.

- 1223 1. If the original QXRI has an existing query component, these parameters MUST be added  
1224 to that query component. (Note that the query parameter names in Table 19 were chosen  
1225 to minimize the probability of collision with any other existing query parameter names.) If  
1226 the original QXRI does not have a query component, one MUST be added to pass these  
1227 parameters. If the original QXRI had a null query component (only a leading question  
1228 mark), or a query component consisting of only question marks, *one additional leading*  
1229 *question mark* MUST be added when adding any XRI resolution parameters. Any XRI  
1230 query parameters and question marks will later be removed by the proxy resolver in the  
1231 URI construction step defined in section 8.4.
- 1232 2. Each parameter MUST be delimited from other parameters by an ampersand (“&”). Any  
1233 occurrences of the character “&” within an input parameter (specifically the Service Type  
1234 value) MUST be percent encoded prior to input.
- 1235 3. Each parameter MUST be delimited from its value by an equals sign (“=”).
- 1236 4. If any parameter name is included but its value is empty, the value of the parameter  
1237 MUST be considered null.
- 1238 5. For proxy resolution, any input parameter supplied explicitly via a QXRI query parameter  
1239 MUST take precedence over the same parameter provided via any other interface, even  
1240 if the value of the QXRI-supplied parameter is explicitly null. See the following section.

## 1241 7.4 HTTP(S) Accept Headers

1242 XRI resolution input parameters MAY also be passed to a proxy resolver via the HTTP(S) Accept  
1243 header of a resolution request. These will take the form of the media type identifiers and  
1244 associated parameters defined in Table 18. Following are the rules governing the use of such  
1245 inputs in XRI proxy resolution.

- 1246 1. If the Accept header consists of multiple media type identifiers and their associated  
1247 parameters as described in section 14.1 of [RFC2616], the proxy resolver MUST choose  
1248 only one to accept. A proxy resolver client SHOULD order media type identifiers  
1249 according to the client’s preference and a proxy resolver SHOULD choose the first one it  
1250 supports.
- 1251 2. If a media type identifier matches one of the Service Types in Table 18, it MUST be  
1252 interpreted as the implicit value of the Resolution Media Type parameter, and the implicit  
1253 value of the Service Media Type parameter MUST be null.
- 1254 3. If a media type identifier does not match one of the Service Types in Table 18, it MUST  
1255 be interpreted as the implicit value of the Service Media Type parameter, and the implicit  
1256 value of the Resolution Media Type parameter MUST be null.
- 1257 4. If the value of the Accept header content type is null, then this MUST be interpreted as  
1258 the implicit value of both the Resolution Media Type and Service Media Type parameters  
1259 (i.e., both are set to null).
- 1260 5. In all cases, if the value of either the Resolution Media Type or Service Media Type  
1261 parameters is explicitly set via the `_xrd_r` or `_xrd_m` query parameters in the QXRI  
1262 (including to a null value), this explicit value MUST override any implicit value that may be  
1263 set via an HTTP(S) Accept header.
- 1264 6. All other rules for processing of the Resolution Media Type input parameter apply from  
1265 section 4.1.2. In particular, the `refs` parameter allows proxy resolver operators to control  
1266 whether their local resolver will perform reference processing, and the `sep` parameter  
1267 allows proxy resolver clients to control whether they want the proxy resolver to perform  
1268 service endpoint selection.

## 1269 **7.5 Null Resolution Media Type**

1270 Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a  
1271 resolution request where the Resolution Media Type input parameter value is null—either  
1272 because this parameter was not implicitly set using an HTTP Accept header media type identifier  
1273 as defined in the previous section or because it was explicitly set to null using the `_xrd_r` query  
1274 parameter.

1275 If the value of the Resolution Media Type value is null, the values of its media type parameters  
1276 MUST be set to `trust=none`, `refs=true`, and `sep=true`. In addition, the output MUST be an  
1277 HTTP(S) redirect as defined in the following section.

## 1278 **7.6 Outputs and HTTP(S) Redirects**

1279 For all values of the Resolution Media Type parameter except null, a proxy resolver MUST follow  
1280 the output rules defined in section 4.2.

1281 If the value of the Resolution Media Type is null, and the output is not an error, a proxy resolver  
1282 MUST follow the rules for output of a URI List as defined in section 4.2.3. However instead of  
1283 returning a URI list, it MUST return the highest priority URI (the first one in the list) as an HTTP(S)  
1284 3XX redirect with the Accept header content type set to the value of the Service Media Type  
1285 parameter.

1286 If the output is an error, a proxy resolver SHOULD return a human-readable error message with a  
1287 media type of either `text/plain` or `text/html`.

1288 This rule enables XRI proxy resolvers to serve clients that do not understand XRI syntax or  
1289 resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the  
1290 service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept  
1291 header (if any).

## 1292 **7.7 Differences Between Proxy Resolution Servers**

1293 An XRI proxy resolution request may be sent to any proxy resolver that will accept it. All XRI  
1294 proxy resolvers SHOULD attempt to deliver uniform responses given the same QXRI and input  
1295 parameters. However because proxy resolvers may potentially need to make decisions about  
1296 network errors, reference processing, and trust policies on behalf of the client they are proxying,  
1297 and these decisions may be based on local policy, in some cases different proxy resolvers may  
1298 return different results.

## 8 Service Endpoint Selection

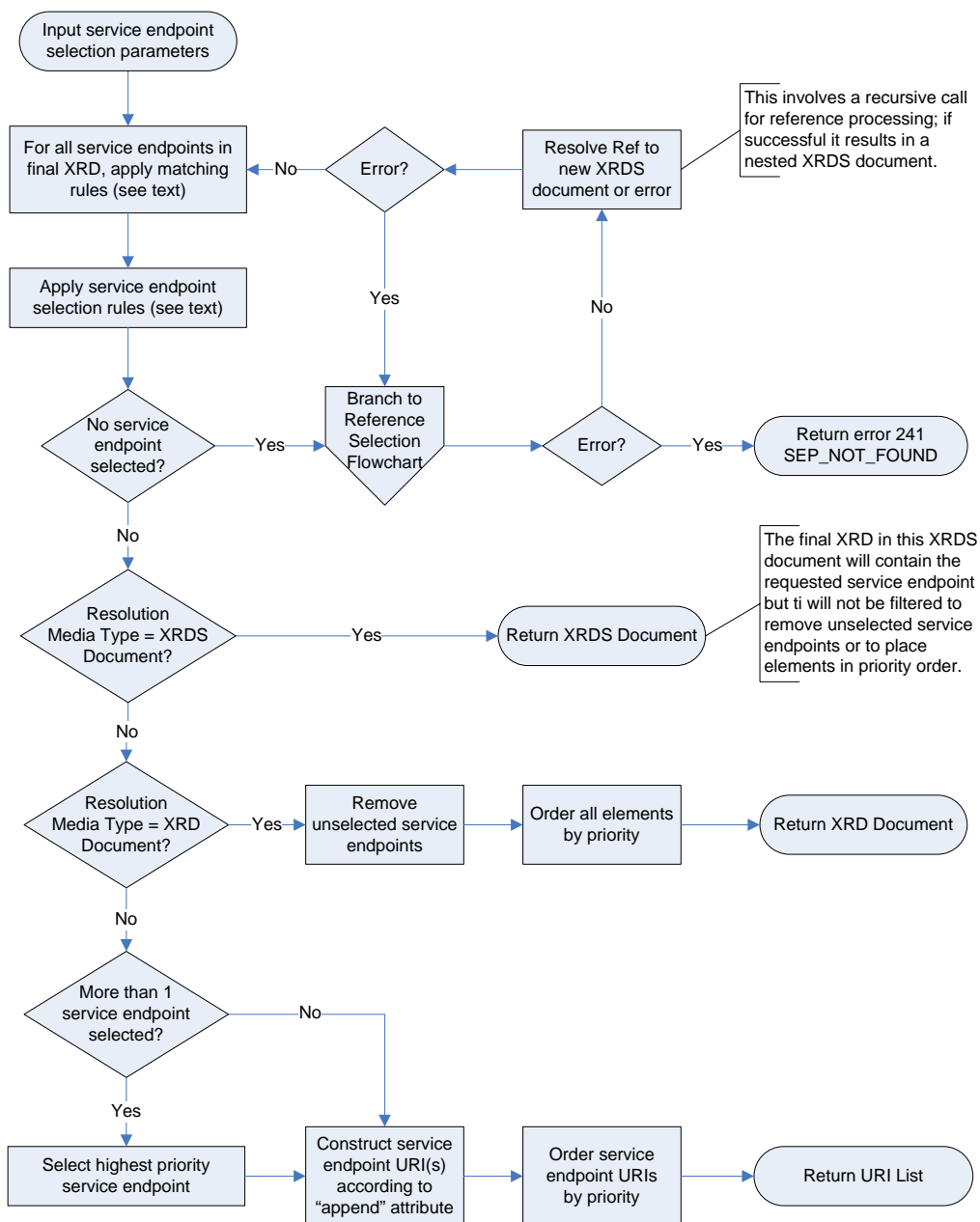
1299

1300 If the authority resolution phase is successful, the output is an XRDS document containing a final  
 1301 XRD describing the target authority. If requested, a resolver may perform an optional second  
 1302 phase of XRI resolution by processing this XRDS document to select a requested service  
 1303 endpoint. This section specifies the rules for this process.

### 8.1 Processing Rules

1304

1305 Figure 3 shows the overall logical flow of the service endpoint selection process.



1306

1307

Figure 3: Service endpoint selection flowchart.

- 1308 Following are the normative rules for the service endpoint selection process:
- 1309 1. The inputs for service endpoint selection include both those defined for authority  
1310 resolution (Table 7) plus those defined only for service endpoint selection (Table 8).
- 1311 2. For each `xrd:Service` element in an XRD, selection is controlled by matching its three  
1312 child service endpoint selection elements (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`)  
1313 to the values of the corresponding the Service Type, Service Media Type, and Path  
1314 String parameters. This matching process MUST follow the rules defined in section 8.2.
- 1315 3. Following the matching process, selection of `xrd:Service` elements MUST follow the  
1316 selection rules defined in section 8.3.
- 1317 4. If after applying the selection rules no service endpoint is selected, a resolver MUST  
1318 perform reference processing as defined in section 9 if the `refs` media type parameter is  
1319 set to `true` or null, or if the parameter is absent. If the `refs` media type parameter is set  
1320 to `false` and the XRD contains at least one `xrd:Ref` element that could be followed,  
1321 the resolver MUST return a successful response with a status code of 101  
1322 `REF_NOT_FOLLOWED`.
- 1323 5. If an XRD does not include the requested service endpoint and does not include any  
1324 `xrd:Ref` elements, the resolver MUST return an error with a status code of 241  
1325 `SEP_NOT_FOUND` regardless of the value of the `refs` media type parameter.
- 1326 6. The output of service endpoint selection MUST be a valid instance of the media type  
1327 value of the Resolution Media Type parameter. If this output is an XRDS Document, an  
1328 XRD Document, or a URI List, it MUST conform to the output requirements defined in  
1329 section 4.2. If the Resolution Media Type value is null and the output is an HTTP(S)  
1330 redirect, the resolver MUST proceed as defined in section 7.6.

## 1331 8.2 Matching Rules

1332 Matching of an `xrd:Service` element is controlled by its three service endpoint selection  
1333 elements, `xrd:Type`, `xrd:MediaType`, and `xrd:Path`, and their optional `xrd:match` attribute.  
1334 Note that the rules in this section only determine *whether there is a match on the service endpoint*  
1335 *selection elements*. The rules for selecting the parent `xrd:Service` element on the basis of  
1336 these matches are defined in section 8.3. For examples, see Appendix H.

### 1337 8.2.1 Match Attribute Values

1338 The `xrd:match` attribute accepts an enumerated list of values as defined in Table 20.

Value	Matching Rule (for corresponding input parameter)
default	Match this element only if there are no other matches in the XRD on this element type (except another default match). This is the default value if an instance of an element that accepts the <code>xrd:match</code> attribute is <i>entirely</i> absent.
content	Match the contents of this element. This is the default value if an element that accepts the <code>xrd:match</code> attribute is present but the attribute is omitted or its value is null.
any	Match any value (null or non-null).
non-null	Match any value except null.
null	Match a null value.
none	Do not match. The containing <code>xrd:Service</code> element is not currently active.



1339 **Table 20: Enumerated values of the global match attribute and corresponding matching rules.**

1340 The following matching rules apply based on the value of the `xrd:match` attribute:

- 1341 1. If an `xrd:Type`, `xrd:MediaType`, or `xrd:Path` element is omitted, it **MUST** be  
1342 considered equivalent to including an instance of the respective element with an  
1343 `xrd:match` attribute value of `default`.
- 1344 2. If an `xrd:Type`, `xrd:MediaType`, or `xrd:Path` element is present but its contents are  
1345 empty, the value of the element **MUST** be considered null.
- 1346 3. If an `xrd:Type`, `xrd:MediaType`, or `xrd:Path` element is present but the `xrd:match`  
1347 attribute is omitted or has the default value of `content`, the respective content matching  
1348 rules in the following three sections **MUST** be applied.
- 1349 4. For all other values of the `xrd:match` attribute, including `default`, `any`, `non-null`,  
1350 `null`, and `none`, the element contents **MUST** be ignored and the matching rules in Table  
1351 20 **MUST** be applied.

## 1352 8.2.2 Service Type Matching

1353 The following rules apply when the `xrd:match` attribute of an `xrd:Type` element is omitted or  
1354 has the default value of `content`.

- 1355 1. The values of the Service Type parameter and the `xrd:Type` element **SHOULD** be  
1356 normalized according to the requirements of their identifier scheme prior to input. In  
1357 addition, if the value is an XRI or an IRI it **MUST** be in URI-normal form as defined in  
1358 section 2.3 of **[XRISyntax]**. XRI resolvers **MAY** perform normalization of these values but  
1359 **MUST NOT** be required to do so.
- 1360 2. To match, the values **MUST** be equivalent according to the equivalence rules of the  
1361 applicable identifier scheme.

## 1362 8.2.3 Service Media Type Matching

1363 The following rules apply when the `xrd:match` attribute of an `xrd:MediaType` element is  
1364 omitted or has the default value of `content`.

- 1365 1. The values of the Service Media Type parameter and the `xrd:MediaType` element  
1366 **SHOULD** be normalized according to the rules for media types in section 3.7 of  
1367 **[RFC2616]** prior to input. (The rules are that type and subtype names are case-  
1368 insensitive, but parameter values may or may not be case-sensitive depending on the  
1369 semantics of the parameter name. XRI resolution media type parameters are case-  
1370 insensitive.) XRI resolvers **MAY** perform normalization of these values but **MUST NOT** be  
1371 required to do so.
- 1372 2. To match, the values **MUST** be character-for-character equivalent.

## 1373 8.2.4 Path String Matching

1374 The following rules apply when the `xrd:match` attribute of an `xrd:Path` element is omitted or  
1375 has the default value of `content`.

- 1376 1. The values of the Path String parameter and the `xrd:Path` element **MUST** be  
1377 normalized to remove any trailing space or XRI delimiter ("`/`", "`**`", or "`!`").
- 1378 2. Equivalence comparison **MUST** be performed using Caseless Matching as defined in  
1379 section 3.13 of **[Unicode]**, with the exception that it is not necessary to perform  
1380 normalization after case folding.
- 1381 3. **IMPORTANT:** If there is no match, this comparison **MUST** be repeated after enclosing  
1382 the value of the Path String parameter in parentheses ("`(`" and "`)`"). This eliminates the

1383 need for XRD authors to specify multiple `xrd:Path` elements in order to match an XRI  
 1384 path that may or may not be expressed as a cross-reference.

1385 4. *IMPORTANT*: If there is still no match, the final subsegment of the Path String parameter  
 1386 and its preceding delimiter **MUST** be removed and the comparison process repeated  
 1387 beginning with step 1. This process **MUST** be repeated until the Path String is null. This  
 1388 enables stem-based path matching.

### 1389 8.3 Selection Rules

1390 After the matching rules in section 8.2 are applied, final selection of `xrd:Service` elements is  
 1391 governed by the value of the `xrd:select` attribute of each matched child element according to  
 1392 the following rules.

- 1393 1. If the `xrd:match` attribute of any child element of an `xrd:Service` element has a value  
 1394 of `none`, then the `xrd:Service` element **MUST NOT** be selected, regardless of the  
 1395 value of any sibling selection element.
- 1396 2. With the exception of the first rule above, If the `xrd:select` attribute of any matched  
 1397 child element of an `xrd:Service` element has a value of `true`, then the `xrd:Service`  
 1398 element **MUST** be selected (i.e., this is an “or” match).
- 1399 3. If the `xrd:select` attribute for all matched child elements of an `xrd:Service` element  
 1400 has a value of `false` or null (the implied value if the attribute is absent), then the  
 1401 `xrd:Service` element **MUST** only be selected if there is a match on at least one  
 1402 `xrd:Type` child element, at least one `xrd:MediaType` child element, and at least one  
 1403 `xrd:Path` child element (i.e., this is an “and” match).

### 1404 8.4 Construction of Service Endpoint URIs

1405 If the output is a URI List, the final step in the service endpoint selection process is construction  
 1406 of the service endpoint URI(s). This is governed by the `xrd:append` attribute of each `xrd:URI`  
 1407 element. The values of this attribute are shown in Table 21.

Value	Component of QXRI to Append
none	None. This is the default for the authority resolution phase. (URI construction for authority resolution service endpoints is defined in section 5.1.7).
local	The entire local part, i.e., one of three cases: a) If only a path is present, append the path (including the leading “/”); b) If only a query is present, append the query (including the leading “?”) c) If both a path and a query are present, append the entire local part (including the leading “/”). This is the default for the service endpoint selection phase.
authority	Authority segment only (including the community root subsegment but no leading delimiter or “xri://”).
path	Path only (including the leading “/”).
query	Query only (including the leading “?”).
qxri	Entire QXRI (including the leading “xri://”).

1408 **Table 21: Values of the append attribute and the corresponding QXRI component to append.**

1409 If the `xrd:append` attribute is empty or omitted, the default value is `none` for the authority  
1410 resolution phase and `local` for the service endpoint selection phase. Following are the rules for  
1411 construction of the final service endpoint URI based on the value of the `xrd:append` attribute.  
1412 *Note that these rules must be followed closely in order to give XRD authors precise control over*  
1413 *construction of service endpoint URIs.*

- 1414 1. If the value is `none`, the exact contents of the `xrd:URI` element MUST be returned  
1415 directly without any further processing.
- 1416 2. For any other value, the exact value of the QXRI component specified in Table 21,  
1417 *including any leading delimiter(s)*, with no additional escaping or percent encoding MUST  
1418 be appended directly to the exact contents of the `xrd:URI` element *including any trailing*  
1419 *delimiter(s)*. If the value of the QXRI component specified in Table 21 consists of only a  
1420 leading delimiter, then this value MUST be appended according to these rules. If the  
1421 value of the QXRI component specified in Table 21 is null, then the contents of the  
1422 `xrd:URI` element MUST be returned directly as if the value of the `xrd:append` attribute  
1423 was `none`.
- 1424 3. If any special query parameters for XRI proxy resolution were added to an existing QXRI  
1425 query component as defined in section 7.3, these query parameters MUST be removed  
1426 prior to performing the append operation. If after removal of these query parameters the  
1427 QXRI query component consists of only *a string of one or more question marks* (the  
1428 delimiting question mark plus zero or more additional question marks) then *exactly one*  
1429 *question mark* MUST also be removed. This preserves the query component of the  
1430 original QXRI if it was null or contained only question marks.

---

## 1431 9 Reference Processing

### 1432 9.1 Synonyms

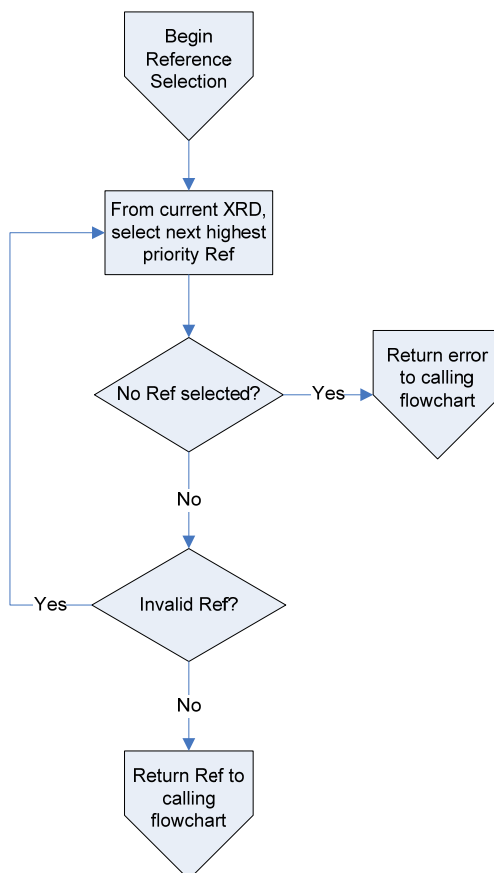
1433 XRI resolution includes support for *synonyms*—XRIs that are not character-for-character  
1434 equivalent, but which identify the same target resource. Three types of synonyms may be  
1435 expressed in XRDs:

- 1436 • *Local synonyms* are expressed using the `xrd:LocalID` element. A local synonym is an  
1437 XRI that is interchangeable with the contents of the `xrd:Query` element. In other words,  
1438 it is an XRI assigned by the same authority and identifying the same target resource as the  
1439 current XRD (this authority is identified by the `xrd:XRD/xrd:ProviderID` element of  
1440 the XRD if present—see section 3.2.) A common example is a persistent XRI assigned to  
1441 a resource that has one or more reassignable XRIs. Resolution of a local synonym **MUST**  
1442 return the same XRD as the one in which the local synonym appears (except for the  
1443 value of the `xrd:Query`, `xrd:Expires`, and `xrd:LocalID` elements).
- 1444 • *Canonical synonyms* are expressed using the `xrd:CanonicalID` element. A canonical  
1445 synonym is the preferred absolute XRI among all absolute XRI synonyms for a resource  
1446 (in the context of the authority providing the current XRD). This XRI may or may not be  
1447 assigned by the same authority providing the current XRD, and if resolvable may or may  
1448 not resolve to the current XRD. For durability, a canonical synonym **SHOULD** be a  
1449 persistent XRI. For simplicity, only one canonical synonym **SHOULD** be assigned to a  
1450 resource, however in certain circumstances (such as merging two previously distinct  
1451 resources into one), this may not be possible.
- 1452 • *References* are expressed using the `xrd:Ref` element. In XRI resolution, a reference is  
1453 an absolute XRI that identifies the same target resource as the query XRI but which  
1454 **MUST** be assigned by a different authority than the authority providing the current XRD.  
1455 A reference **MAY** resolve to a different XRD than the current XRD, i.e., an XRD  
1456 containing different synonyms, service endpoints, or other metadata describing the target  
1457 resource. (Note that such an XRD **MAY** also include a reference back to the current  
1458 QXRI, in which case the references will be circular.)

1459 Of these three synonym types, references play a special role in XRI resolution. In both authority  
1460 resolution and service endpoint selection, if the current XRD does not contain a requested  
1461 `xrd:Service` element but contains at least one `xrd:Ref` element, a resolver may follow these  
1462 reference(s) according to the rules defined in this section.

1463 **9.2 Processing Rules**

1464 Figure 4 is an overview of the logical flow of selecting a reference for processing.



1465

1466

**Figure 4: Flowchart for selecting an XRI reference for processing.**

1467 Following are the normative rules that apply to reference processing:

- 1468
- 1469
- 1470
- 1471
- 1472
- 1473
- 1474
- 1475
- 1476
- 1477
- 1478
- 1479
- 1480
- 1481
- 1482
- 1483
1. The resolver **MUST** begin by selecting the highest priority `xrd:Ref` element in the current XRD. If not found, this is an error and the resolver **MUST** proceed as defined in section 5 for authority resolution or section 8 for service endpoint selection.
  2. The contents of the selected `xrd:Ref` element **MUST** be a valid QXRI as defined in section 4.1.1. If not, it **MUST** be ignored and the step above repeated to select the next highest priority reference.
  3. Once a valid reference has been selected, the resolver **MUST** begin resolution of a new XRDS document beginning with the community root authority of the reference XRI as defined in section 5.1.3. The resolver **MUST** use the same resolution input parameters as for the original QXRI. For reference processing to complete successfully, the resolver **MUST** complete resolution of the entire Authority String of the reference XRI (including following any further references if necessary). If the reference XRI includes a Path String or Query String (including any resolution query parameters), they **MUST** be ignored.
  4. If reference processing is successful and the Resolution Media Type is an XRDS document, the XRDS document resulting from reference processing **MUST** be nested inside the parent XRDS document as defined in section 9.3.

- 1484 5. If reference processing is successful and the Resolution Media Type is an XRD  
1485 document, the output **MUST** be the final XRD document returned.
- 1486 6. If reference processing is successful and the Resolution Media Type is a URI List or an  
1487 HTTP(S) redirect, it **MUST** be based on the final XRD document returned.

### 1488 **9.3 Nesting XRDS Documents**

1489 If a reference is followed successfully, it will produce an XRDS document that fully describes the  
1490 reference. This XRDS document **MUST** be included in the containing XRDS document  
1491 immediately following the `xrd:XRD` element that contains the `xrd:Ref` element being followed.  
1492 In addition, the `xrds:XRDS/@xrds:ref` attribute of this nested XRDS document **MUST** be set  
1493 to the value of the `xrd:Ref` element it describes.

1494 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the  
1495 original QXRI even if resolution traverses references. Note that nested XRDS documents do not  
1496 include an XRD for the community root subsegment because this is part of the configuration of  
1497 the resolver.

1498 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD whose  
1499 `xml:id` attribute value matches the `xml:id` attribute value of any previous XRD in the chain of  
1500 resolution requests beginning with the original QXRI, the resolver **MUST** replace this XRD with an  
1501 empty XRD element. The resolver **MUST** set this empty element's `xrd:idref` attribute value to  
1502 the value of the `xrd:XRD/xml:id` attribute of the matched XRD element. This prevents  
1503 conflicting `xrd:XRD/xml:id` values.

1504 In the following example the original query XRI is `xri://@a*b*c`. The XRD for `xri://@a*b`  
1505 does not contain an authority resolution service endpoint but includes a reference to  
1506 `xri://@x*y`. The elements and attributes specific to reference processing are shown in bold.

```

1507 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
1508   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
1509     <Query>*a</Query>
1510     ...
1511     <Service>
1512       <Type>xri://$res*auth*($v*2.0)</Type>
1513       <URI>http://a.example.com</URI>
1514     </Service>
1515   </XRD>
1516   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
1517     <Query>*b</Query>
1518     ...
1519     <Ref>xri://@x*y</Ref>
1520     <Service>
1521       ...no authority resolution service endpoint...
1522     </Service>
1523   </XRD>
1524   <XRDS ref="xri://@x*y">
1525     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
1526       <Query>*x</Query>
1527       ...
1528       <Service>
1529         <Type>xri://$res*auth*($v*2.0)</Type>
1530         <URI>http://x.example.com</URI>
1531       </Service>
1532     </XRD>
1533     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
1534       <Query>*y</Query>
1535       ...
1536       <Service>
1537         <Type>xri://$res*auth*($v*2.0)</Type>
1538         <URI>http://y.example.com</URI>
1539       </Service>
1540     </XRD>
1541   </XRDS>
1542   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
1543     <Query>*c</Query>
1544     ...
1545     <Service>
1546       ...final service endpoints described here...
1547     </Service>
1548   </XRD>
1549 </XRDS>

```

1550

## 10 Error Processing

1551

### 10.1 Error Codes

1552 XRI resolution error codes are patterned after the HTTP model. They are broken into three major  
1553 categories:

- 1554
- 1xx: Success—the requested resolution operation was completed successfully.
  - 1555 • 2xx: Permanent errors—the resolver encountered an error from which it could not  
1556 recover.
  - 1557 • 3xx: Temporary errors—the resolver encountered an error condition that may be only  
1558 temporary.

1559 Each major category is broken into five minor categories:

- 1560
- x0x: General error that may take place during any phase of resolution.
  - 1561 • x1x: Input error.
  - 1562 • x2x: Generic authority resolution error.
  - 1563 • x3x: Trusted authority resolution error.
  - 1564 • x4x: Service endpoint (SEP) selection error.

1565 The full list of XRI resolution error codes is defined in Table 22.

1566

Code	Symbolic Error	Phase(s)	Description
100	SUCCESS	Any	Operation was successful.
101	REF_NOT_FOLLOWED	Any	Operation was successful to the point where a reference needed to be processed, but the resolver was instructed by the <code>refs</code> parameter not to follow references.
200	PERM_FAIL	Any	Generic permanent failure.
201	NOT_IMPLEMENTED	Any	The requested function (trusted resolution, service endpoint selection) is not implement by the resolver.
202	LIMIT_EXCEEDED	Any	A locally configured resource limit was exceeded. Examples: number of references to follow, number of XRD elements that can be handled, size of an XRDS document.
210	INVALID_INPUT	Input	Generic input error.
211	INVALID_QXRI	Input	Input QXRI does not conform to XRI syntax.
212	INVALID_RES_MEDIA_TYPE	Input	Input Resolution Media Type is invalid.
213	INVALID_SEP_TYPE	Input	Input Service Type is invalid.
214	INVALID_SEP_MEDIA_TYPE	Input	Input Service Media Type is invalid.



215	UNKNOWN_ROOT	Input	Community root specified in QXRI is not configured in the resolver.
220	AUTH_RES_ERROR	Authority resolution	Generic authority resolution error
221	AUTH_RES_NOT_FOUND	Authority resolution	The subsegment cannot be resolved due to a missing authority resolution service in an XRD.
222	QUERY_NOT_FOUND	Authority resolution	Responding authority does not have an XRI matching the query.
223	UNEXPECTED_XRD	Authority resolution	Value of the <code>xrd:Query</code> element does not match the subsegment requested.
230	TRUSTED_RES_ERROR	Trusted resolution	Generic trusted resolution error
231	HTTPS_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS authority resolution endpoint.
232	SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate a SAML authority resolution endpoint.
233	HTTPS+SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS+SAML authority resolution endpoint.
234	UNVERIFIED_SIGNATURE	Trusted resolution	Signature verification failed.
240	SEP_SELECTION_ERROR	SEP selection	Generic service endpoint selection error
241	SEP_NOT_FOUND	SEP selection	The requested service endpoint could not be found in the current XRD or via reference processing.
300	TEMPORARY_FAIL	Any	Generic temporary failure.
301	TIMEOUT_ERROR	Any	Locally-defined timeout limit has lapsed during an operation (e.g. network latency).
320	NETWORK_ERROR	Authority resolution	Generic error during authority resolution phase (includes uncaught exception, system error, network error).
321	UNEXPECTED_RESPONSE	Authority resolution	When querying an authority resolution service, the server returned a non-200 HTTP status.
322	INVALID_XRDS	Authority resolution	Invalid XRDS received from an authority resolution service (includes malformed XML, truncated content, or wrong content type).

Table 22: Error codes for XRI resolution.

1568 As defined in section 4.2, when resolution output is an error and the output format is an XRDS  
1569 Document or XRD Document, the error code is returned as the value of the `xrd:code` attribute  
1570 of the `xrd:Status` element. When the resolution output is a URI List, the error code is returned  
1571 as the first line of a plain text message.

## 1572 **10.2 Error Context Strings**

1573 Each error code in Table 22 may be returned with an optional error context string that provides  
1574 additional human-readable information about the error. When resolution output is an XRDS  
1575 Document or XRD Document, this string is returned as the contents of the `xrd:Status` element.  
1576 When the resolution output is a URI List, this string is returned as the second line of a plain text  
1577 message. Implementers SHOULD provide error context strings with additional information about  
1578 an error and possible solutions whenever it can be helpful to developers or end users.

## 1579 **10.3 Error Handling in Recursing and Proxy Resolution**

1580 In recursing and proxy resolution (sections 5.1.5 and 7), a server is acting as a client resolver for  
1581 other authority resolution service endpoints. If in this intermediary capacity it receives an  
1582 unrecoverable error, it MUST return the error to the originating client in the output format  
1583 specified by the value of the requested Resolution Media Type as defined in section 4.2.

1584 If the output format is an XRDS Document, it MUST contain `xrd:XRD` elements for all  
1585 subsegments successfully resolved or retrieved from cache prior to the error. The final `xrd:XRD`  
1586 element MUST include the `xrd:Query` element that produced the error and the `xrd:Status`  
1587 element that describes the error as defined above.

1588 If the output format is an XRD Document, it MUST include the `xrd:Query` element that produced  
1589 the error and the `xrd:Status` element that describes the error as defined above.

1590 If this output format is a URI List, it MUST be returned with the content type `text/plain`. The  
1591 first line MUST consist of only the numeric error code as defined in section 10.1 followed by a  
1592 CRLF. The second line is OPTIONAL; if present it MUST be the error context string as defined in  
1593 section 10.2.

1594 If the value of the Resolution Media Type is null (which can only happen in proxy resolution as  
1595 described in section 7.5), rather than returning an HTTP(S) redirect, a proxy resolver SHOULD  
1596 return a human-readable error message with a media type of either `text/plain` or `text/html`.  
1597 It is particularly important in this case to return an error message that will be understandable to an  
1598 end-user who may have no understanding of XRI resolution or the fact that the error is coming  
1599 from an XRI proxy resolver.

---

## 1600 11 Use of HTTP(S)

### 1601 11.1 HTTP Errors

1602 When a resolver encounters fatal HTTP(S) errors during the resolution process, it MUST return  
1603 the appropriate XRI resolution error code and error message as defined in section 10. In this way  
1604 calling applications do not have to deal separately with XRI and HTTP error messages.

### 1605 11.2 HTTP Headers

#### 1606 11.2.1 Caching

1607 The HTTP caching capabilities described by [RFC2616] should be leveraged for all types of XRI  
1608 resolution service. Specifically, implementations SHOULD implement the caching model  
1609 described in section 13 of [RFC2616], and in particular, the “Expiration Model” of section 13.2, as  
1610 this requires the fewest round-trip network connections.

1611 All XRI resolution servers SHOULD send the Cache-Control or Expires headers in their  
1612 responses per section 13.2 of [RFC2616] unless there are overriding security or policy reasons to  
1613 omit them.

1614 Note that HTTP Cache headers SHOULD NOT conflict with expiration information in an XRD.  
1615 That is, the expiration date specified by HTTP caching headers SHOULD NOT be later than any  
1616 of the expiration dates for any of the `xrd:Expires` elements returned in the HTTP response.  
1617 This implies that recursing and proxy resolvers SHOULD compute the “soonest” expiration date  
1618 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching  
1619 headers for the HTTP response.

#### 1620 11.2.2 Location

1621 During HTTP interaction, “Location” headers may be present per [RFC2616] (i.e., during 3XX  
1622 redirects). Redirects SHOULD be made cacheable through appropriate HTTP headers, as  
1623 specified in section 11.2.1.

#### 1624 11.2.3 Content-Type

1625 For authority resolution, the “Content-type” header in the 2XX responses MUST contain the  
1626 media type identifier values specified in Table 11 (for generic resolution), Table 15 (for HTTPS  
1627 trusted resolution), Table 16 (for SAML trusted resolution), or Table 17 (or HTTPS+SAML trusted  
1628 resolution).

1629 Following service endpoint selection, clients and servers MAY negotiate content type using  
1630 standard HTTP content negotiation features. Regardless of whether this feature is used,  
1631 however, the server MUST respond with an appropriate media type in the “Content-type” header  
1632 if the resource is found and an appropriate content type is returned.

### 1633 11.3 Other HTTP Features

1634 HTTP provides a number of other features including transfer-coding, proxying, validation-model  
1635 caching, and so forth. All these features may be used insofar as they do not conflict with the  
1636 required uses of HTTP described in this document.

## 1637 **11.4 Caching and Efficiency**

1638 In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the  
1639 application level. For best results, however, resolution clients SHOULD be conservative with  
1640 caching expiration semantics, including cache expiration dates. This implies that in a series of  
1641 HTTP redirects, for example, the results of the entire process SHOULD only be cached as long  
1642 as the shortest period of time allowed by any of the intermediate HTTP responses.

1643 Because not all HTTP client libraries expose caching expiration to applications, identifier  
1644 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the  
1645 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments  
1646 should be mindful of limitations in current HTTP clients and proxies.

1647 The cache expiration time of an XRD may also be explicitly limited by the identifier authority. If the  
1648 expiration time in the `xrd:Expires` element is sooner than the expiration time calculated from  
1649 the HTTP caching semantics, the XRD MUST be discarded before the expiration time in  
1650 `xrd:Expires`. Note also that a `saml:Assertion` element returned during SAML trusted  
1651 resolution has its own signature expiration semantics as defined in **[SAML]**. While this may  
1652 invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if  
1653 it is not expired by HTTP caching semantics or the `xrd:Expires` element.

1654 With both application-level and HTTP-level caching, the resolution process is designed to have  
1655 minimal overhead. Resolution of each qualified subsegment of an XRI authority segment is a  
1656 separate step described by a separate XRD, so intermediate results can typically be cached in  
1657 their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified  
1658 subsegments, which are common to more identifiers, will naturally result in a greater number of  
1659 cache hits than resolution of lower-level subsegments.

---

## 1660 12 Extensibility and Versioning

### 1661 12.1 Extensibility

#### 1662 12.1.1 Extensibility of XRDs

1663 The XRD schema in Appendix A use an an open-content model that is designed to be extended  
1664 with other metadata. In most places, extension elements and attributes from namespaces other  
1665 than `xri://$xrd*($v*2.0)` are explicitly allowed. These extension points are designed to  
1666 simplify default processing using a “Must Ignore” rule. The base rule is that unrecognized  
1667 elements and attributes, and the content and child elements of unrecognized elements, MUST be  
1668 ignored. As a consequence, elements that would normally be recognized by a processor MUST  
1669 be ignored if they appear as descendants of an unrecognized element.

1670 Extension elements MUST NOT require new interpretation of elements defined in this document.  
1671 If an extension element is present, a processor MUST be able to ignore it and still correctly  
1672 process the XRD document.

1673 Extension specifications MAY simulate “Must Understand” behavior by applying an “enclosure”  
1674 pattern. Elements defined by the XRD schema in Appendix A whose meaning or interpretation is  
1675 modified by extension elements can be wrapped in a extension container element defined by the  
1676 extension specification. This extension container element SHOULD be in the same namespace  
1677 as the other extension elements defined by the extension specification.

1678 Using this design, all elements whose interpretations are modified by the extension will now be  
1679 contained in the extension container element and thus will be ignored by clients or other  
1680 applications unable to process the extension. The following example illustrates this pattern using  
1681 an extension container element from an extension namespace (`other:SuperService`) that  
1682 contains an extension element (`other:ExtensionElement`):

```
1683 <XRD>  
1684   <Service>  
1685     ...  
1686   </Service>  
1687   <other:SuperService>  
1688     <Service>  
1689       ...  
1690       <other:ExtensionElement>...</other:ExtensionElement>  
1691     </Service>  
1692   </other:SuperService>  
1693 </XRD>
```

1694 In this example, the `other:ExtensionElement` modifies the interpretation or processing rules  
1695 for the parent `xrd:Service` element and therefore must be understood by the consumer for the  
1696 proper interpretation of the parent `xrd:Service` element. To preserve the correct interpretation  
1697 of the `xrd:Service` element in this context, the `xrd:Service` element is “wrapped” so only  
1698 consumers that understand elements in the `other:SuperService` namespace will attempt to  
1699 process the `xrd:ProviderID` element.

1700 The addition of extension elements does not change the requirement for SAML signatures to be  
1701 verified across all elements, whether recognized or not.

#### 1702 12.1.2 Other Points of Extensibility

1703 The use of HTTP, XML, XRI, and URIs in the design of XRDS documents, XRD elements, and  
1704 XRI resolution architecture provides additional specific points of extensibility:

- 1705 • Specification of new resolution service types or other service types using XRI or URIs as  
1706 values of the `xrd:Type` element.
- 1707 • Specification of new resolution output formats or features using media types and media  
1708 type parameters as values of the `xrd:MediaType` element as defined in [RFC2045] and  
1709 [RFC2046].
- 1710 • HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- 1711 • Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- 1712 • Use of cross-references within XRIs, particularly for associating new types of metadata  
1713 with a resource. See [XRIMetadata].

## 1714 12.2 Versioning

1715 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,  
1716 this section describes versioning guidelines.

### 1717 12.2.1 Version Numbering

1718 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number  
1719 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version  
1720 number *MajorA.MinorA* if and only if:

1721  $Major_B > Major_A$  OR ( (  $Major_B = Major_A$  ) AND  $Minor_B > Minor_A$  )

### 1722 12.2.2 Versioning of the XRI Resolution Specification

1723 New releases of the XRI Resolution specification may specify changes to the resolution protocols  
1724 and/or the XRD schema in Appendix A. When changes affect either of these, the resolution  
1725 service type version number will be changed. Where changes are purely editorial, the version  
1726 number will not be changed.

1727 In general, if a change is backward-compatible, the new version will be identified using the  
1728 current major version number and a new minor version number. If the change is not backward-  
1729 compatible, the new version will be identified with a new major version number.

### 1730 12.2.3 Versioning of XRDs

1731 The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have  
1732 no specific knowledge of the elements it may contain. Therefore it has no version element, and  
1733 can remain stable indefinitely because there is no need to version its namespace.

1734 The `xrd:XRD` element has an optional `xrd:version` attribute. When used, the value of this  
1735 attribute MUST be the exact numeric version value of the XRI Resolution specification to which its  
1736 containing elements conform.

1737 When new versions of the XRI Resolution specification are released, the namespace for the XRD  
1738 schema may or may not be changed. If there is a major version number change, the namespace  
1739 for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the  
1740 namespace for the `xrd:XRD` schema may remain unchanged.

1741 With regard to versioning, this specification follows the same guidelines as established in section  
1742 4.2.1 of [SAML]:

1743 *In general, maintaining namespace stability while adding or changing the content of a*  
1744 *schema are competing goals. While certain design strategies can facilitate such changes,*  
1745 *it is complex to predict how older implementations will react to any given change, making*  
1746 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*  
1747 *minor revisions is reserved, in the interest of namespace stability. Except in special*

1748 *circumstances (for example, to correct major deficiencies or to fix errors),*  
1749 *implementations should expect forward-compatible schema changes in minor revisions,*  
1750 *allowing new messages to validate against older schemas.*

1751 *Implementations SHOULD expect and be prepared to deal with new extensions and*  
1752 *message types in accordance with the processing rules laid out for those types. Minor*  
1753 *revisions MAY introduce new types that leverage the extension facilities described in [this*  
1754 *section]. Older implementations SHOULD reject such extensions gracefully when they*  
1755 *are encountered in contexts that dictate mandatory semantics.*

#### 1756 **12.2.4 Versioning of Protocols**

1757 The protocols defined in this document may also be versioned by future releases of the XRI  
1758 Resolution specification. If these protocols are not backward-compatible with older  
1759 implementations, they will be assigned a new XRI with a new version identifier for use in  
1760 identifying their service type in XRDs. See section 2.1.2.

1761 Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP  
1762 provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an  
1763 XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely  
1764 to continue to use the same XRI to identify the protocol as was used in previous versions of the  
1765 XRI Resolution specification.

---

## 1766 **13 Security and Data Protection**

1767 Significant portions of this specification deal directly with security issues, and these will not be  
1768 summarized again here. In addition, basic security practices and typical risks in resolution  
1769 protocols are well-documented in many other specifications. Only security considerations directly  
1770 relevant to XRI resolution are included here.

### 1771 **13.1 DNS Spoofing or Poisoning**

1772 When XRI resolution is deployed to use HTTP URIs or other URIs which include DNS names, the  
1773 accuracy of the XRI resolution response may be dependent on the accuracy of DNS queries. For  
1774 those deployments where DNS is not trusted, the resolution infrastructure may be deployed with  
1775 HTTP URIs that use IP addresses in the authority portion of HTTP URIs and/or with the trusted  
1776 resolution mechanisms defined by this specification. Resolution results obtained using trusted  
1777 resolution can be evaluated independently of DNS resolution results. While this does not solve  
1778 the problem of DNS spoofing, it does allow the client to detect an error condition and reject the  
1779 resolution result as untrustworthy. In addition, **[DNSSEC]** may be considered if DNS names are  
1780 used in HTTP URIs.

### 1781 **13.2 HTTP Security**

1782 Many of the security considerations set forth in HTTP/1.1 **[RFC2616]** apply to XRI Resolution  
1783 protocols defined here. In particular, confidentiality of the communication channel is not  
1784 guaranteed by HTTP. Server-authenticated HTTPS should be used in cases where confidentiality  
1785 of resolution requests and responses is desired.

1786 Special consideration should be given to proxy and caching behaviors to ensure accurate and  
1787 reliable responses from resolution requests. For various reasons, network topologies increasingly  
1788 have transparent proxies, some of which may insert VIA and other headers as a consequence, or  
1789 may even cache content without regard to caching policies set by a resource's HTTP authority.

1790 Implementations of XRI Proxies and caching authorities should also take special note of the  
1791 security recommendations in HTTP/1.1 **[RFC2616]** section 15.7

### 1792 **13.3 SAML Considerations**

1793 SAML trusted authority resolution must adhere to the rules defined by the SAML 2.0 Core  
1794 Specification **[SAML]**. Particularly noteworthy are the XML Transform restrictions on XML  
1795 Signature and the enforcement of the SAML Conditions element regarding the validity period.

### 1796 **13.4 Limitations of Trusted Resolution**

1797 While the trusted resolution protocols specified in this document provides a way to verify the  
1798 integrity of a successful XRI resolution, it may not provide a way to verify the integrity of a  
1799 resolution failure. Reasons for this limitation include the prevalence of non-malicious network  
1800 failures, the existence of denial-of-service attacks, and the ability of a man-in-the-middle attacker  
1801 to modify HTTP responses when resolution is not performed over HTTPS.

1802 Additionally, there is no revocation mechanism for the keys used in trusted resolution. Therefore,  
1803 a signed resolution's validity period should be limited appropriately to mitigate the risk of an  
1804 incorrect or invalid resolution.



## 1805 **13.5 Community Root Authorities**

1806 The XRI authority information for a community root needs to be well-known to the clients that  
1807 request resolution within that community. For trusted resolution, this includes the authority  
1808 resolution service endpoint URIs, the `xrd:XRD/xrd:ProviderID`, and the `ds:KeyInfo`  
1809 information. An acceptable means of providing this information is for the community root authority  
1810 to produce a self-signed XRD and publish it to a server-authenticated HTTPS endpoint. Special  
1811 care should be taken to ensure the correctness of such an XRD; if this information is incorrect, an  
1812 attacker may be able to convince a client of an incorrect result during trusted resolution.

## 1813 **13.6 Caching Authorities**

1814 In addition to traditional HTTP caching proxies, XRI proxy resolvers may be a part of the  
1815 resolution topology. Such proxy resolvers should take special precautions against cache  
1816 poisoning, as these caching entities may represent trusted decision points within a deployment's  
1817 resolution architecture.

## 1818 **13.7 Recursing and Proxy Resolution**

1819 During recursing resolution, subsegments of the XRI authority segment for which the resolving  
1820 network endpoint is not authoritative may be revealed to that service endpoint. During proxy  
1821 resolution, some or all of an XRI is provided to the proxy resolver.

1822 In both cases, privacy considerations should be evaluated before disclosing such information.

## 1823 **13.8 Denial-Of-Service Attacks**

1824 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks  
1825 typical of systems relying on DNS and HTTP.

# 14References

## 1827 14.1 Normative

- 1828 [DNSSEC] D. Eastlake, *Domain Name System Security Extensions*,  
1829 <http://www.ietf.org/rfc/rfc2535>, RFC 2535, March 1999.
- 1830 [RFC2045] N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*  
1831 *Part One: Format of Internet Message Bodies*,  
1832 <http://www.ietf.org/rfc/rfc2045.txt>, RFC 2045, November 1996.
- 1833 [RFC2046] N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*  
1834 *Part Two: Media Types*, <http://www.ietf.org/rfc/rfc2046.txt>, RFC 2046,  
1835 November 1996.
- 1836 [RFC2119] S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*,  
1837 <http://www.ietf.org/rfc/rfc2119.txt>, RFC 2119, March 1997.
- 1838 [RFC2141] R. Moats, *URN Syntax*, <http://www.ietf.org/rfc/rfc2141.txt>, IETF RFC  
1839 2141, May 1997.
- 1840 [RFC2234] D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications:*  
1841 *ABNF*, <http://www.ietf.org/rfc/rfc2234.txt>, RFC 2234, November 1997.
- 1842 [RFC2483] M. Mealling, R. Daniel Jr., *URI Resolution Services Necessary for URN*  
1843 *Resolution*, <http://www.ietf.org/rfc/rfc2483.txt>, RFC 2483, January 1999.
- 1844 [RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.  
1845 Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*,  
1846 <http://www.ietf.org/rfc/rfc2616.txt>, RFC 2616, June 1999.
- 1847 [RFC2818] E. Rescorla, *HTTP over TLS*, <http://www.ietf.org/rfc/rfc2818.txt>, RFC  
1848 2818, May 2000.
- 1849 [SAML] S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for*  
1850 *the OASIS Security Assertion Markup Language (SAML) V2.0*,  
1851 <http://www.oasis-open.org/committees/security>, March 2005.
- 1852 [Unicode] The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined  
1853 by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley,  
1854 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1  
1855 (<http://www.unicode.org/versions/Unicode4.0.1>) and by Unicode 4.1.0  
1856 (<http://www.unicode.org/versions/Unicode4.1.0>), March, 2005.
- 1857 [UUID] Open Systems Interconnection – *Remote Procedure Call*, ISO/IEC  
1858 11578:1996, <http://www.iso.org/>, August 2001.
- 1859 [XML] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau,  
1860 *Extensible Markup Language (XML) 1.0, Third Edition*, World Wide Web  
1861 Consortium, <http://www.w3.org/TR/REC-xml/>, February 2004.
- 1862 [XMLDSig] D. Eastlake, J. Reagle, D. Solo et al., *XML-Signature Syntax and*  
1863 *Processing*, World Wide Web Consortium,  
1864 <http://www.w3.org/TR/xmlsig-core/>, February, 2002.
- 1865 [XMLID] J. Marsh, D. Veillard, N. Walsh, *xml:id Version 1.0*, World Wide Web  
1866 Consortium, <http://www.w3.org/TR/2005/REC-xml-id-20050909>,  
1867 September 2005.
- 1868 [XMLSchema] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, *XML Schema Part*  
1869 *1: Structures Second Edition*, World Wide Web Consortium,  
1870 <http://www.w3.org/TR/xmlschema-1/>, October 2004.

- 1871 **[XMLSchema2]** P. Biron, A. Malhotra, *XML Schema Part 2: Datatypes Second Edition*,  
1872 World Wide Web Consortium, <http://www.w3.org/TR/xmlschema-2/>,  
1873 October 2004.
- 1874 **[XRIMetadata]** D. Reed, *Extensible Resource Identifier (XRI) Metadata V2.0*,  
1875 <http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf>,  
1876 March 2005.
- 1877 **[XRISyntax]** D. Reed, D. McAlpin, *Extensible Resource Identifier (XRI) Syntax V2.0*,  
1878 <http://docs.oasis-open.org/xri/xri/V2.0/xri-syntax-V2.0-cd-01.pdf>, March  
1879 2005.

## 1880 **14.2 Informative**

- 1881 **[XRIFAQ]** OASIS XRI Technical Committee, *XRI 2.0 FAQ*, [http://www.oasis-](http://www.oasis-open.org/committees/xri/faq.php)  
1882 [open.org/committees/xri/faq.php](http://www.oasis-open.org/committees/xri/faq.php), Work-In-Progress, March 2006.
- 1883 **[XRIReqs]** G. Wachob, D. Reed, M. Le Maitre, D. McAlpin, D. McPherson,  
1884 *Extensible Resource Identifier (XRI) Requirements and Glossary v1.0*,  
1885 [http://www.oasis-](http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc)  
1886 [open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-](http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc)  
1887 [and-glossary-v1.0.doc](http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc), June 2003.

1888

## Appendix A. XML Schema for XRDS and XRD (Normative)

1889

```
1890 <?xml version="1.0" encoding="UTF-8"?>
1891 <xs:schema targetNamespace="xri://$xrds" elementFormDefault="qualified"
1892 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds">
1893   <!-- Utility patterns -->
1894   <xs:attributeGroup name="otherattribute">
1895     <xs:anyAttribute namespace="##other" processContents="lax"/>
1896   </xs:attributeGroup>
1897   <xs:group name="otherelement">
1898     <xs:choice>
1899       <xs:any namespace="##other" processContents="lax"/>
1900       <xs:any namespace="##local" processContents="lax"/>
1901     </xs:choice>
1902   </xs:group>
1903   <!-- Patterns for elements -->
1904   <xs:element name="XRDS">
1905     <xs:complexType>
1906       <xs:sequence>
1907         <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
1908       </xs:sequence>
1909       <xs:attributeGroup ref="xrds:otherattribute"/>
1910       <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
1911     </xs:complexType>
1912   </xs:element>
1913 </xs:schema>
1914
1915
1916 <?xml version="1.0" encoding="UTF-8"?>
1917 <xs:schema targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified"
1918 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1919 xmlns:xrd="xri://$xrd*($v*2.0)">
1920   <!-- Utility patterns -->
1921   <xs:attributeGroup name="otherattribute">
1922     <xs:anyAttribute namespace="##other" processContents="lax"/>
1923   </xs:attributeGroup>
1924   <xs:group name="otherelement">
1925     <xs:choice>
1926       <xs:any namespace="##other" processContents="lax"/>
1927       <xs:any namespace="##local" processContents="lax"/>
1928     </xs:choice>
1929   </xs:group>
1930   <xs:attributeGroup name="priorityAttrGrp">
1931     <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
1932   </xs:attributeGroup>
1933   <xs:attributeGroup name="selectionAttrGrp">
1934     <xs:attribute name="match" use="optional" default="default">
1935       <xs:simpleType>
1936         <xs:restriction base="xs:string">
1937           <xs:enumeration value="default"/>
1938           <xs:enumeration value="content"/>
1939           <xs:enumeration value="any"/>
1940           <xs:enumeration value="non-null"/>
1941           <xs:enumeration value="null"/>
1942           <xs:enumeration value="none"/>
1943         </xs:restriction>
1944       </xs:simpleType>
1945     </xs:attribute>
1946     <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
1947   </xs:attributeGroup>
1948   <xs:complexType name="URIPattern">
1949     <xs:simpleContent>
1950       <xs:extension base="xs:anyURI">
1951         <xs:attributeGroup ref="xrd:otherattribute"/>
1952       </xs:extension>

```

```

1953     </xs:simpleContent>
1954 </xs:complexType>
1955 <xs:complexType name="URIPriorityPattern">
1956   <xs:simpleContent>
1957     <xs:extension base="xrd:URIPattern">
1958       <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
1959     </xs:extension>
1960   </xs:simpleContent>
1961 </xs:complexType>
1962 <xs:complexType name="StringPattern">
1963   <xs:simpleContent>
1964     <xs:extension base="xs:string">
1965       <xs:attributeGroup ref="xrd:otherattribute"/>
1966     </xs:extension>
1967   </xs:simpleContent>
1968 </xs:complexType>
1969 <xs:complexType name="StringSelectionPattern">
1970   <xs:simpleContent>
1971     <xs:extension base="xrd:StringPattern">
1972       <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
1973     </xs:extension>
1974   </xs:simpleContent>
1975 </xs:complexType>
1976 <!-- Patterns for elements -->
1977 <xs:element name="XRD">
1978   <xs:complexType>
1979     <xs:sequence>
1980       <xs:element ref="xrd:Query" minOccurs="0"/>
1981       <xs:element ref="xrd:Status" minOccurs="0"/>
1982       <xs:element ref="xrd:Expires" minOccurs="0"/>
1983       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
1984       <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
1985       <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded"/>
1986       <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
1987       <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded"/>
1988       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
1989     </xs:sequence>
1990     <xs:attribute name="id" type="xs:ID"/>
1991     <xs:attribute name="idref" type="xs:IDREF" use="optional"/>
1992     <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0"/>
1993     <xs:attributeGroup ref="xrd:otherattribute"/>
1994   </xs:complexType>
1995 </xs:element>
1996 <xs:element name="Query" type="xrd:StringPattern"/>
1997 <xs:element name="Status">
1998   <xs:complexType>
1999     <xs:simpleContent>
2000       <xs:extension base="xrd:StringPattern">
2001         <xs:attribute name="code" type="xs:int" use="required"/>
2002         <xs:attributeGroup ref="xrd:otherattribute"/>
2003       </xs:extension>
2004     </xs:simpleContent>
2005   </xs:complexType>
2006 </xs:element>
2007 <xs:element name="Expires">
2008   <xs:complexType>
2009     <xs:simpleContent>
2010       <xs:extension base="xs:dateTime">
2011         <xs:attributeGroup ref="xrd:otherattribute"/>
2012       </xs:extension>
2013     </xs:simpleContent>
2014   </xs:complexType>
2015 </xs:element>
2016 <xs:element name="ProviderID" type="xrd:URIPattern"/>
2017 <xs:element name="LocalID">
2018   <xs:complexType>
2019     <xs:simpleContent>
2020       <xs:extension base="xrd:StringPattern">
2021         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2022       </xs:extension>
2023     </xs:simpleContent>

```

```

2024     </xs:complexType>
2025 </xs:element>
2026 <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>
2027 <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
2028 <xs:element name="Service">
2029   <xs:complexType>
2030     <xs:sequence>
2031       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
2032       <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
2033       <xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
2034       <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
2035       <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
2036       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2037     </xs:sequence>
2038     <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2039     <xs:attributeGroup ref="xrd:otherattribute"/>
2040   </xs:complexType>
2041 </xs:element>
2042 <xs:element name="Type">
2043   <xs:complexType>
2044     <xs:simpleContent>
2045       <xs:extension base="xrd:URIPattern">
2046         <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
2047       </xs:extension>
2048     </xs:simpleContent>
2049   </xs:complexType>
2050 </xs:element>
2051 <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
2052 <xs:element name="Path" type="xrd:StringSelectionPattern"/>
2053 <xs:element name="URI">
2054   <xs:complexType>
2055     <xs:simpleContent>
2056       <xs:extension base="xrd:URIPattern">
2057         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2058         <xs:attribute name="append">
2059           <xs:simpleType>
2060             <xs:restriction base="xs:string">
2061               <xs:enumeration value="none"/>
2062               <xs:enumeration value="local"/>
2063               <xs:enumeration value="authority"/>
2064               <xs:enumeration value="path"/>
2065               <xs:enumeration value="query"/>
2066               <xs:enumeration value="qxri"/>
2067             </xs:restriction>
2068           </xs:simpleType>
2069         </xs:attribute>
2070       </xs:extension>
2071     </xs:simpleContent>
2072   </xs:complexType>
2073 </xs:element>
2074 </xs:schema>
2075
2076

```

---

2077 **Appendix B. RelaxNG Compact Syntax Schema**  
2078 **for XRDS and XRD (Informative)**

2079 [TODO]

---

2080 **Appendix C. Media Type Definition for**  
2081 **application/xrds+xml (Normative)**

2082 [TODO]



---

2083 **Appendix D. Media Type Definition for**  
2084 **application/xrd+xml (Normative)**

2085 [TODO]

2086 **Appendix E. Example Local Resolver Interface**  
 2087 **Definition (Informative)**

2088 Following is a language-neutral example of an interface definition for a local XRI resolver  
 2089 consistent with the requirements of this specification.

2090 The interface definition is provided as five operations where each operation takes three or more  
 2091 of the following input parameters. The input parameters are described here in terms of the  
 2092 normative text in section 4. In all of these parameters, the value empty string ("") is interpreted the  
 2093 same as the value null.

2094

Parameter name	Description
QXRI	Query XRI as defined in section 4.1.1.
trustType	The value of the <code>trust</code> media type subparameter of the Resolution Media Type parameter as specified in Table 6 of section 2.3, whose behavior is defined in section 4.1.2.
followRefs	The value of the <code>refs</code> media type subparameter of the Resolution Media Type parameter as specified in Table 6 of section 2.3, whose behavior is defined in section 4.1.2.
sepType	Service Type as defined in Table 8 of section 4.1.
sepMediaType	Service Media Type as defined in Table 8 of section 4.1.

2095

2096 The five operations correspond to the following combinations of values of the Resolution Media  
 2097 Type parameter and its `sep` (service endpoint) subparameter (section 4.1.2) as shown below.

2098

	Operation name	Resolution Media Type Parameter Value	sep Subparameter Value
1	<code>resolveAuthToXRDS</code>	<code>application/xrds+xml</code>	<code>false</code>
2	<code>resolveAuthToXRD</code>	<code>application/xrd+xml</code>	<code>false</code>
3	<code>resolveSepToXRDS</code>	<code>application/xrds+xml</code>	<code>true</code>
4	<code>resolveSepToXRD</code>	<code>application/xrd+xml</code>	<code>true</code>
5	<code>resolveSepToURIList</code>	<code>text/uri-list</code>	<code>ignored</code>

2099 Following is the API and descriptions of the five operations.

## 2100 1. Resolve Authority to XRDS

```
2101 int resolveAuthToXRDS(  
2102     in string QXRI, in string trustType, in boolean followRefs,  
2103     out string XRDS, out string errorContext);
```

- 2104 • Performs authority resolution only (sections 5 and 6) and outputs the XRDS as specified  
2105 in section 4.2.1 when the `sep` subparameter is `false`.
- 2106 • Only the authority segment of the QXRI is processed by this function. If the QXRI  
2107 contains a path or query component, it is ignored.
- 2108 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 2109 • Returns the error code. If error, then the `errorContext` output argument may contain  
2110 additional error information. The output XRDS will contain a final XRD with the same  
2111 status code and optional context information in its `xrd:Status` element.

2112

## 2113 2. Resolve Authority to XRD

```
2114 int resolveAuthToXRD(  
2115     in string QXRI, in string trustType, in boolean followRefs,  
2116     out string XRD, out string errorContext);
```

- 2117 • Performs authority resolution only (sections 5 and 6) and outputs the final XRD as  
2118 specified in section 4.2.2 when the `sep` subparameter is `false`.
- 2119 • Only the authority segment of the QXRI is processed by this function. If the QXRI  
2120 contains a path or query component, it is ignored.
- 2121 • The output XRD argument will be signed or not depending on the value of `trustType`.
- 2122 • Returns the error code. If error, then the `errorContext` output argument may contain  
2123 additional error information. The output XRD will contain the same status code and  
2124 optional context information in its `xrd:Status` element.

2125

### 2126 3. Resolve Service Endpoint to XRDS

```
2127 int resolveSEPToXRDS(  
2128     in string QXRI, in string trustType, in string sepType,  
2129     in string sepMediaType, in boolean followRefs,  
2130     out string XRDS, out string errorContext);
```

- 2131 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section  
2132 8) and outputs the XRDS as specified in section 4.2.1 when the `sep` subparameter is  
2133 `true`.
- 2134 • As specified in section 4.2.1, the output XRDS document will contain a nested XRDS  
2135 document as the final child element if reference processing was necessary to locate the  
2136 request service endpoint and the `followRefs` flag was set to `true`.
- 2137 • The final XRD in the output XRD will either contain at least one instance of the requested  
2138 service endpoint or an error.
- 2139 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 2140 • Returns the error code. If error, then the `errorContext` output argument may contain  
2141 additional error information. The output XRDS will contain a final XRD with the same  
2142 status code and optional context information in its `xrd:Status` element.
- 2143 • For parameters `sepType` and `sepMediaType`, the value empty string ("" ) is interpreted  
2144 the same as the value `null`.

### 2145 4. Resolve Service Endpoint to XRD

```
2147 int resolveSEPToXRD(  
2148     in string QXRI, in string trustType, in string sepType,  
2149     in string sepMediaType, in boolean followRefs,  
2150     out string XRDS, out string errorContext);
```

- 2151 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section  
2152 8) and outputs the XRD as specified in section 4.2.2 when the `sep` subparameter is  
2153 `true`.
- 2154 • As specified in section 4.2.2, all elements in the output XRD subject to the global  
2155 `xrd:priority` attribute will be returned in order of highest to lowest.
- 2156 • The output XRD will either contain at least one instance of the requested service  
2157 endpoint or an error.
- 2158 • The output XRD will be *not* be signed regardless of the value of `trustType` because the  
2159 XRD will be filtered to only the selected service endpoints.
- 2160 • Returns the error code. If error, then the `errorContext` output argument may contain  
2161 additional error information. The output XRD will contain the same status code and  
2162 optional context information in its `xrd:Status` element.
- 2163 • For parameters `sepType` and `sepMediaType`, the value empty string ("" ) is interpreted  
2164 the same as the value `null`.

2165 **5. Resolve Service Endpoint to URI List**

```
2166 int resolveSepToURIList(  
2167     in string QXRI, in string trustType, in string sepType,  
2168     in string sepMediaType, in boolean followRefs,  
2169     out string[] URIList, out string errorContext);
```

- 2170
- 2171
- Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8) and, upon success, outputs a non-empty URI List as specified in section 4.2.3.
- 2172
- Returns the error code. If error, then the output URI List will be empty, and the `errorContext` output argument may contain additional error information.
- 2173
- For parameters `sepType` and `sepMediaType`, the value empty string ("" ) is interpreted the same as the value null.
- 2174
- 2175
- 2176

---

2177 **Appendix F. Examples of Generic Authority**  
2178 **Resolution (Informative)**

2179 [TODO]

---

2180 **Appendix G. Examples of SAML Trusted Authority**  
2181 **Resolution (Informative)**

2182 [TODO]

---

2183 **Appendix H. Examples of Service Endpoint**  
2184 **Selection (Informative)**

2185 [TODO]



2186

---

## Appendix I. Acknowledgments

2187 The editors would like to acknowledge the contributions of the OASIS XRI Technical Committee,  
2188 whose voting members at the time of publication were:

- 2189 • Geoffrey Strongin, Advanced Micro Devices
- 2190 • Ajay Madhok, AmSoft Systems
- 2191 • Jean-Jacques Dubray, Attachmate
- 2192 • William Barnhill, Booz Allen and Hamilton
- 2193 • Drummond Reed, Cordance Corporation
- 2194 • Marc Le Maitre, Cordance Corporation
- 2195 • Dave McAlpin, Epok
- 2196 • Loren West, Epok
- 2197 • Peter Davis, NeuStar
- 2198 • Masaki Nishitani, Nomura Research
- 2199 • Nat Sakimura, Nomura Research
- 2200 • Tetsu Watanabe, Nomura Research
- 2201 • Owen Davis, PlaNetwork
- 2202 • Victor Grey, PlaNetwork
- 2203 • Fen Labalme, PlaNetwork
- 2204 • Mike Lindelsee, Visa International
- 2205 • Gabriel Wachob, Visa International
- 2206 • Dave Wentker, Visa International
- 2207 • Bill Washburn, XDI.ORG

2208 The editors also would like to acknowledge the following people for their contributions to previous  
2209 versions of the OASIS XRI specifications (affiliations listed for OASIS members):

- 2210 Thomas Bikeev, EAN International; Krishna Sankar, Cisco; Winston Bumpus, Dell; Joseph  
2211 Moeller, EDS; Steve Green, Epok; Lance Hood, Epok; Adarbad Master, Epok; Davis McPherson,  
2212 Epok; Chetan Sabnis, Epok; Phillipe LeBlanc, GemPlus; Jim Schreckengast, Gemplus; Xavier  
2213 Serret, Gemplus; John McGarvey, IBM; Reva Modi, Infosys; Krishnan Rajagopalan, Novell;  
2214 Tomonori Seki, NRI; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,  
2215 Verisign; Rajeev Maria, Visa International; Terence Spielman, Visa International; John Veizades,  
2216 Visa International; Lark Allen, Wave Systems; Michael Willett, Wave Systems; Matthew Dovey;  
2217 Eamonn Neylon; Mary Nishikawa; Lars Marius Garshol; Norman Paskin; Bernard Vatant.
- 2218 • A special acknowledgement to Jerry Kindall (Epok) for a full editorial review.

---

2219

## Appendix J. Notices

2220 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
2221 that might be claimed to pertain to the implementation or use of the technology described in this  
2222 document or the extent to which any license under such rights might or might not be available;  
2223 neither does it represent that it has made any effort to identify any such rights.

2224 Information on OASIS's procedures with respect to rights in OASIS specifications can be found at  
2225 the OASIS website. Copies of claims of rights made available for publication and any assurances  
2226 of licenses to be made available, or the result of an attempt made to obtain a general license or  
2227 permission for the use of such proprietary rights by implementors or users of this specification,  
2228 can be obtained from the OASIS President.

2229 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
2230 applications, or other proprietary rights which may cover technology that may be required to  
2231 implement this specification. Please address the information to the OASIS President.

2232 **Copyright © OASIS Open 2006. All Rights Reserved.**

2233 This document and translations of it may be copied and furnished to others, and derivative works  
2234 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
2235 published and distributed, in whole or in part, without restriction of any kind, provided that the  
2236 above copyright notice and this paragraph are included on all such copies and derivative works.  
2237 However, this document itself does not be modified in any way, such as by removing the  
2238 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS  
2239 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual  
2240 Property Rights document must be followed, or as required to translate it into languages other  
2241 than English.

2242 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
2243 successors or assigns.

2244 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
2245 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
2246 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE  
2247 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
2248 PARTICULAR PURPOSE.