



Extensible Resource Identifier (XRI) Resolution Version 2.0

Working Draft 11, ED 03

24 July 2007

Specification URIs:

This Version:

<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0-wd-11.html>
<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0-wd-11.pdf>
<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0-wd-11.doc>

Previous Version:

<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0-wd-10.html>
<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0-wd-10.pdf>
<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0-wd-10.doc>

Latest Version:

<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0.html>
<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0.pdf>
<http://docs.oasis-open.org/xri/xri/V2.0/xri-resolution-V2.0.doc>

Latest Approved Version:

[http://docs.oasis-open.org/xri/xri/V2.0/\[additional path/filename\].html](http://docs.oasis-open.org/xri/xri/V2.0/[additional path/filename].html)
[http://docs.oasis-open.org/xri/xri/V2.0/\[additional path/filename\].pdf](http://docs.oasis-open.org/xri/xri/V2.0/[additional path/filename].pdf)
[\[http://docs.oasis-open.org/xri/xri/V2.0/\[additional path/filename\].doc\]](http://docs.oasis-open.org/xri/xri/V2.0/[additional path/filename].doc)

Technical Committee:

OASIS eXtensible Resource Identifier (XRI) TC

Chairs:

Gabe Wachob, AmSoft <gabe.wachob@amsoft.net>
Drummond Reed, Cordance <drummond.reed@cordance.net>

Editors:

Gabe Wachob, AmSoft <gabe.wachob@amsoft.net>
Drummond Reed, Cordance <drummond.reed@cordance.net>
Les Chasen, NeuStar <les.chasen@neustar.biz>
William Tan, NeuStar <william.tan@neustar.biz>
Steve Churchill, XDI.org <steven.churchill@xdi.org>

Contributors:

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

Comment [DSR1]: TODO: All URIs need review

Comment [LR2]: This section is not in the template. Also, will need to update the information in this section. Check guide.

Comment [DSR3]: Need to check status of OASIS membership.

Comment [DSR4]: Status?

Related Work:

Comment [DSR5]: TODO

This specification replaces or supercedes:

- [specifications replaced by this standard]
- [specifications replaced by this standard]

This specification is related to:

- [related specifications]
- [related specifications]

Declared XML Namespace(s)

xri://\$res
xri://\$xrds
xri://\$xrd
xri://\$xrd*(\$v*2.0)
xri://\$res*auth
xri://\$res*auth*(\$v*2.0)
xri://\$res*proxy
xri://\$res*proxy*(\$v*2.0)

Abstract:

This document defines both generic and trusted HTTP(S)-based resolution protocols for Extensible Resource Identifiers (XRI) as defined by *Extensible Resource Identifier (XRI) Syntax V2.0 [XRISyntax]* or higher. For a dictionary of XRI defined to provide standardized identifier metadata, see *Extensible Resource Identifier (XRI) Metadata V2.0 [XRIMetadata]*. For a basic introduction to XRI, 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 “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

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>.

Notices

Comment [LR6]: TODO: review to ensure it is correct.

Copyright © OASIS® 1993–2007. All Rights Reserved. OASIS trademark, IPR and other policies apply. All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

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

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

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Comment [DSR7]: TODO - Also check and see if there should be any indication in this statement of our RF IPR mode.

Table of Contents

1	Introduction.....	8
1.1	Overview of XRI Resolution Architecture.....	8
1.2	Structure of this Specification.....	10
1.3	Examples of XRI Resolution Requests and Responses.....	11
1.4	Terminology and Notation.....	11
1.5	Normative References.....	12
1.6	Non-Normative References.....	13
2	Conformance.....	14
2.1	Conformance Targets.....	14
2.2	XRI Resolvers.....	14
2.2.1	Local Resolvers.....	14
2.2.2	Proxy Resolvers.....	14
2.3	XRI Authority Servers.....	14
3	Namespaces.....	15
3.1	XRI Namespaces for XRI Resolution.....	15
3.1.1	XRIs Reserved for XRI Resolution.....	15
3.1.2	XRIs Assigned to XRI Resolution Service Types.....	15
3.2	XML Namespaces for XRI Resolution.....	15
3.3	Media Types for XRI Resolution.....	16
4	XRDS Documents.....	18
4.1	XRDS and XRD Namespaces.....	18
4.2	XRD Elements and Attributes.....	18
4.2.1	Management Elements.....	20
4.2.2	Authority Trust Elements.....	20
4.2.3	Authority Synonym Elements.....	21
4.2.4	Service Endpoint Management Elements.....	21
4.2.5	Service Endpoint Synonym Elements.....	22
4.2.6	Service Endpoint Trust Elements.....	22
4.2.7	Service Endpoint Selection Elements.....	23
4.3	XRD Attribute Processing Rules.....	23
4.3.1	ID Attribute.....	23
4.3.2	Version Attribute.....	23
4.3.3	Priority Attribute.....	24
4.4	XRI and IRI Encoding Requirements.....	24
5	Discovering an XRDS Document from an HTTP(S) URI.....	25
5.1	Overview.....	25
5.2	Protocol Using an HTTP HEAD Request.....	25
5.3	Protocol Using an HTTP GET Request.....	25
6	XRI Resolution Inputs and Outputs.....	27
6.1	Inputs.....	27
6.1.1	QXRI (Authority String, Path String, and Query String).....	28
6.1.2	Resolution Output Format.....	28

6.1.3 Service Type	29
6.1.4 Service Media Type	29
6.2 Outputs	30
6.2.1 XRDS Document.....	30
6.2.2 XRD Document	30
6.2.3 URI List.....	31
6.2.4 HTTP(S) Redirect	31
7 Generic Authority Resolution	32
7.1 XRI Authority Resolution	32
7.1.1 Service Type and Service Media Type.....	32
7.1.2 Protocol.....	33
7.1.3 Community Root Authorities	35
7.1.4 Self-Describing XRDS Documents.....	35
7.1.5 Qualified Subsegments	36
7.1.6 Cross-References	37
7.1.7 Recursing Authority Resolution.....	37
7.1.8 Construction of the Next Authority URI	38
7.2 IRI Authority Resolution.....	38
7.2.1 Service Type and Media Type	39
7.2.2 Protocol.....	39
7.2.3 Optional Use of HTTPS.....	39
8 Trusted Authority Resolution.....	40
8.1 HTTPS	40
8.1.1 Service Type and Service Media Type.....	40
8.1.2 Protocol.....	40
8.2 SAML	40
8.2.1 Service Type and Service Media Type.....	41
8.2.2 Protocol.....	41
8.2.3 Recursing Authority Resolution	42
8.2.4 Client Validation of XRDS.....	42
8.2.5 Correlation of ProviderID and KeyInfo Elements	43
8.3 HTTPS+SAML	44
8.3.1 Service Type and Service Media Type.....	44
8.3.2 Protocol.....	44
9 Proxy Resolution.....	45
9.1 Service Type and Media Types	45
9.2 HXRIs.....	45
9.3 HXRI Query Parameters	47
9.4 HTTP(S) Accept Headers.....	48
9.5 Null Resolution Output Format	48
9.6 Outputs and HTTP(S) Redirects.....	48
9.7 Differences Between Proxy Resolution Servers	49
9.8 Combining Authority and Proxy Resolution Servers	49
10 Service Endpoint Selection	50
10.1 Processing Rules	50

10.2	Service Endpoint Selection Logic	52
10.3	Selection Element Matching Rules	53
10.3.1	Selection Element Match Options	54
10.3.2	The Match Attribute	54
10.3.3	Absent Selection Element Matching Rule	54
10.3.4	Empty Selection Element Matching Rule	55
10.3.5	Multiple Selection Element Matching Rule	55
10.3.6	Type Element Matching Rules	55
10.3.7	Path Element Matching Rules	55
10.3.8	MediaType Element Matching Rules	56
10.4	Service Endpoint Matching Rules	56
10.4.1	Service Endpoint Match Options	56
10.4.2	Select Attribute Match Rule	56
10.4.3	All Positive Match Rule	56
10.4.4	Default Match Rule	56
10.5	Service Endpoint Selection Rules	57
10.5.1	Positive Match Rule	57
10.5.2	Default Match Rule	57
10.6	Pseudocode	57
10.7	Construction of Service Endpoint URIs	58
11	Synonyms	60
11.1	Hierarchical and Polyarchical Synonyms	60
11.2	LocalID	62
11.3	GlobalID	63
11.4	Ref (Forward Reference)	63
11.5	Backref (Backward Reference)	64
11.6	CanonicalID	64
12	Synonym Verification	65
12.1	Hierarchical Verification Rules	65
12.1.1	HTTP(S) URIs	65
12.1.2	XRIs	65
12.2	Polyarchical Verification Rules	66
12.2.1	HTTP(S) URIs	66
12.2.2	XRIs	66
12.3	Hierarchical Verification Examples	66
12.3.1	HTTP(S) URI to HTTP(S) URI	66
12.3.2	XRI to XRI	66
12.4	Polyarchical Verification Examples	67
12.4.1	HTTP(S) URI to HTTP(S) URI	67
12.4.2	HTTP(S) URI to XRI	68
12.4.3	XRI to HTTP(S) URI	68
12.4.4	XRI to XRI	69
13	Reference Processing	71
13.1	Authority References	71
13.1.1	Processing Rules	71

13.1.2 Nesting XRDS Documents	72
13.2 Service References	73
13.2.1 Processing Rules	73
13.2.2 Adding XRD Documents	75
14 Error Processing	77
14.1 Error Codes	77
14.2 Error Context Strings	79
14.3 Error Handling in Recursing and Proxy Resolution	79
15 Use of HTTP(S)	80
15.1 HTTP Errors	80
15.2 HTTP Headers	80
15.2.1 Caching	80
15.2.2 Location	80
15.2.3 Content-Type	80
15.3 Other HTTP Features	80
15.4 Caching and Efficiency	81
15.4.1 Resolver Caching	81
15.4.2 Synonyms	81
16 Extensibility and Versioning	82
16.1 Extensibility	82
16.1.1 Extensibility of XRDS	82
16.1.2 Other Points of Extensibility	82
16.2 Versioning	83
16.2.1 Version Numbering	83
16.2.2 Versioning of the XRI Resolution Specification	83
16.2.3 Versioning of XRDS	83
16.2.4 Versioning of Protocols	84
17 Security and Data Protection	85
17.1 DNS Spoofing or Poisoning	85
17.2 HTTP Security	85
17.3 SAML Considerations	85
17.4 Limitations of Trusted Resolution	85
17.5 Community Root Authorities	86
17.6 Caching Authorities	86
17.7 Recursing and Proxy Resolution	86
17.8 Denial-Of-Service Attacks	86
A. Acknowledgments	87
B. Non-Normative Text	88
C. Revision History	89
D. XML Schema for XRDS and XRD (Normative)	90
E. RelaxNG Compact Syntax Schema for XRDS and XRD (Informative)	93
F. Media Type Definition for application/xrds+xml (Normative)	94
G. Media Type Definition for application/xrd+xml (Normative)	95
H. Example Local Resolver Interface Definition (Informative)	96

1 Introduction

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

1.1 Overview of XRI Resolution Architecture

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

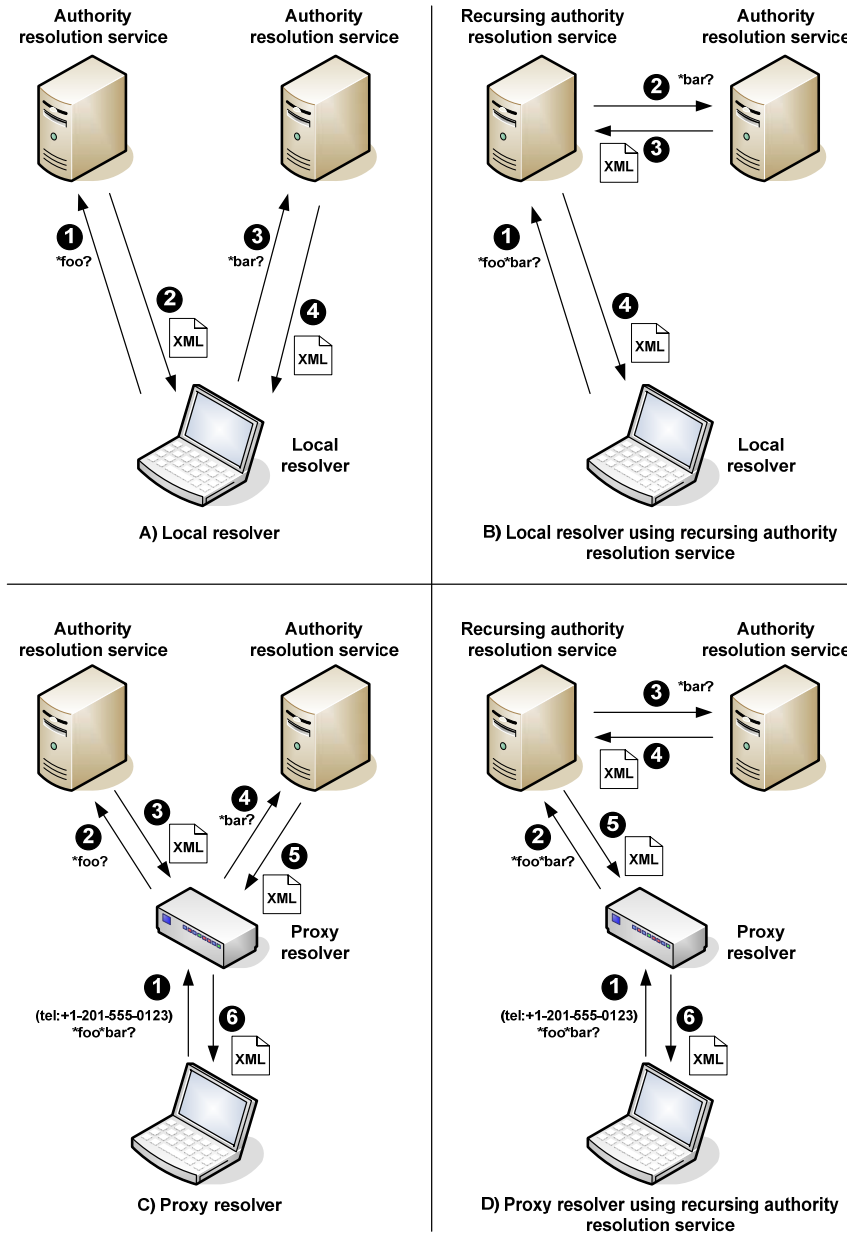
Non-recurring resolvers rely on *recursing nameservers* to do this work. For example, a non-recurring resolver might query a recurring nameserver for the entire DNS name `docs.oasis-open.org`. The nameserver would then do the job of querying the `org` nameserver for the IP address of `oasis-open`, then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

XRI resolution follows this same architecture except at a higher level of abstraction, i.e., rather than using UDP to resolve a domain name into an attribute of a text-based resource descriptor, it uses HTTP(S) to resolve an XRI into an XML-based resource descriptor called an *XRDS document*. Table 1 provides a high-level 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, Global, Canonical, Ref, Backref
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
Recurring resolution	recursing nameserver	recursing authority resolution service or proxy resolver

Table 1: Comparing DNS and XRI resolution architecture.

29 As Table 1 notes, XRI resolution architecture supports both recursing authority resolution
 30 services and *proxy resolvers*. A proxy resolver is simply a local XRI resolver with an HTTP(S)
 31 interface. Proxy resolvers enable applications—even those that do not natively understand XRIs
 32 but can process HTTP URIs—to access the functions of an XRI resolver remotely.
 33 Figure 1 shows four scenarios of how these components might interact to resolve
 34 `xri://(tel:+1-201-555-0123)*foo*bar` (note that, unlike DNS, this works from left-to-
 35 right).



36
 37 Figure 1: Four typical scenarios for XRI authority resolution.

38 In each of these scenarios, two phases of XRI resolution may be involved:

- 39 • *Phase 1: Authority Resolution.* This is the phase required to resolve the authority segment of
40 an XRI into an XRDS document describing the target authority. Authority resolution works
41 iteratively from left-to-right across each subsegment in the authority segment of the XRI (in
42 XRI, subsegments are delimited using either a specified set of symbol characters or
43 parentheses). For example, in the XRI `xri://(http://example.root)*foo*bar`, the
44 authority subsegments are `(http://example.root)` (the community root authority, in this
45 case a URI expressed as an cross-reference delimited with parentheses), `*foo`, (the first
46 resolvable subsegment), and `*bar`, (the second resolvable subsegment). Note that a
47 resolver must be preconfigured (or have its own way of discovering) the community root
48 authority starting point, so the community root subsegment is not resolved except in one
49 special case (see [ref to new section about self-descriptions]).
- 50 • *Phase 2: Service Endpoint Selection.* Once authority resolution is complete, the optional
51 second phase of XRI resolution is to select a specific set of metadata from the final XRDS
52 document retrieved. Although an XRDS document may contain any type of metadata
53 describing the target resource, this specification defines a ruleset for selecting *service*
54 *endpoints*: descriptors of concrete URIs at which network services are available for the target
55 resource. An XRI resolver may optionally use the path and/or query components of an XRI to
56 select the service endpoint(s) to return to a calling application.

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

- 58 • *HTTP.* As a resolution protocol, HTTP not only makes it easy to deploy XRI resolution
59 services (including proxy resolution services), but allows them to employ both HTTP security
60 standards (e.g., HTTPS) and XML-based security standards (e.g., SAML). Although less
61 efficient than UDP, HTTP(S) is suitable for the higher level of abstraction represented by
62 XRI and can take advantage of the full caching capabilities of modern web servers.
- 63 • *XRDS documents.* This simple, extensible XML resource description format makes it easy to
64 describe the capabilities of any XRI-, IRI-, or URI-identified resource in a manner that can be
65 consumed by any XML-aware application.
- 66 • *Synonyms and cross-references.* DNS uses the CNAME attribute to establish equivalence
67 between domain names. XRDS documents include five synonym elements (LocalID,
68 GlobalID, CanonicalID, Ref, and Backref) to provide robust support for mapping XRI, IRI, or
69 URIs to other XRI, IRI, or URIs that represent the same resource. This is particularly useful
70 for discovering and mapping persistent identifiers often required by trust infrastructures. The
71 use of XRI cross-references also enables multiple authorities to maintain distributed XRDS
72 documents describing the same logical resource.
- 73 • *Service endpoint descriptors.* DNS can use NAPTR records to do string transformations into
74 URIs representing network endpoints. XRDS documents have *service endpoint descriptors*—
75 elements that describe the set of URIs at which a particular type of service is available. Each
76 service endpoint may present a different type of data or metadata representing or describing
77 the identified resource. Thus XRI resolution can serve as a lightweight, interoperable
78 discovery mechanism for resource attributes available via HTTP(S), LDAP, UDDI, SAML,
79 WS-Trust, or other directory or discovery protocols.

80 1.2 Structure of this Specification

81 This specification is structured into the following major sections:

- 82 • *Conformance* (section 2) specifies the conformance targets and conformance claims for this
83 specification.
- 84 • *Namespaces* (section 3) specifies the XRI and XML namespaces and media types used for
85 the XRI resolution protocol.

- 86 • *XRDS Documents* (section 4) specifies a simple, flexible XML-based container for XRI
87 resolution metadata and/or other metadata describing a resource.
- 88 • *Discovering an XRDS Document from an HTTP(S) URI* (section 5) specifies how to obtain
89 XRDS metadata describing a resource, including XRI synonyms for that resource, starting
90 from an HTTP(S) URI identifying that resource.
- 91 • *Inputs and Outputs* (section 6) specifies the standard input parameters and output formats for
92 XRI resolution.
- 93 • *Generic Authority Resolution* (section 7) specifies a simple resolution protocol for the
94 authority segment of an XRI using HTTP/HTTPS as a transport.
- 95 • *Trusted Authority Resolution* (section 8) specifies three extensions to generic authority
96 resolution for creating a chain of trust between the participating identifier authorities using
97 HTTPS connections, SAML assertions, or both.
- 98 • *Proxy Resolution* (section 9) specifies an HTTP(S) interface for an XRI resolver plus a format
99 for expressing an XRI as an HTTP(S) URI to provide backwards compatibility with existing
100 HTTP(S) infrastructure.
- 101 • *Service Endpoint Selection* (section 10) specifies an optional second phase of resolution for
102 selecting a set of service endpoints from an XRDS document.
- 103 • *Synonyms* (section 11) specifies the usage rules for the set of synonym elements supported
104 in XRDS documents.
- 105 • *Synonym Verification* (section 12) specifies how a resolver can verify that one XRI, IRI, or
106 URI is an authorized canonical synonym for another.
- 107 • *Reference Processing* (section 13) specifies how a resolver follows a Ref synonym from one
108 XRDS document to another to enable federation of XRDS documents across multiple
109 identifier authorities.
- 110 • *Error Processing* (section 14) specifies error codes and error handling.
- 111 • *Use of HTTP(S)* (section 15) specifies how the XRI resolution protocol leverages features of
112 the HTTP(S) protocol.
- 113 • *Extensibility and Versioning* (section 16) describes how the XRI resolution protocol can be
114 easily extended and how new versions will be identified and accommodated.
- 115 • *Security and Data Protection* (section 17) summarizes key security and privacy
116 considerations for XRI resolution infrastructure.

117 **1.3 Examples of XRI Resolution Requests and Responses**

118 To minimize non-normative material in the main body of the specification, examples of XRI
119 resolution requests and responses are compiled in a separate non-normative document from the
120 XRI TC, *XRI 2.0 Implementers Guide* [**XRIGuide**].

121 **1.4 Terminology and Notation**

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

126 This specification uses the Augmented Backus-Naur Form (ABNF) syntax notation defined in
127 [**RFC4234**].

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

130 Formatting conventions used in this document:

131 Examples look like this.

132 ABNF productions look like this.

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

134 1.5 Normative References

- 135 **[DNSSEC]** D. Eastlake, *Domain Name System Security Extensions*,
136 <http://www.ietf.org/rfc/rfc2535>, IETF RFC 2535, March 1999.
- 137 **[RFC2045]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
138 *Part One: Format of Internet Message Bodies*,
139 <http://www.ietf.org/rfc/rfc2045.txt>, IETF RFC 2045, November 1996.
- 140 **[RFC2046]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
141 *Part Two: Media Types*, <http://www.ietf.org/rfc/rfc2046.txt>, IETF RFC
142 2046, November 1996.
- 143 **[RFC2119]** S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*,
144 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 145 **[RFC2141]** R. Moats, *URN Syntax*, <http://www.ietf.org/rfc/rfc2141.txt>, IETF RFC
146 2141, May 1997.
- 147 **[RFC2483]** M. Mealling, R. Daniel Jr., *URI Resolution Services Necessary for URN*
148 *Resolution*, <http://www.ietf.org/rfc/rfc2483.txt>, IETF RFC 2483, January
149 1999.
- 150 **[RFC2616]** R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.
151 Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*,
152 <http://www.ietf.org/rfc/rfc2616.txt>, IETF RFC 2616, June 1999.
- 153 **[RFC2818]** E. Rescorla, *HTTP over TLS*, <http://www.ietf.org/rfc/rfc2818.txt>, IETF
154 RFC 2818, May 2000.
- 155 **[RFC3023]** M. Murata, S. St-Laurent, D. Kohn, *XML Media Types*,
156 <http://www.ietf.org/rfc/rfc3023.txt>, IETF RFC 3023, January 2001.
- 157 **[RFC4234]** D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications:*
158 *ABNF*, <http://www.ietf.org/rfc/rfc4234.txt>, IETF RFC 4234, October 2005.
- 159 **[RFC4288]** N. Freed, J. Klensin, *Media Type Specifications and Registration*
160 *Procedures*, <http://www.ietf.org/rfc/rfc4288.txt>, IETF RFC 4288,
161 December 2005.
- 162 **[SAML]** S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for*
163 *the OASIS Security Assertion Markup Language (SAML) V2.0*,
164 <http://www.oasis-open.org/committees/security>, March 2005.
- 165 **[Unicode]** The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined
166 by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley,
167 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1
168 (<http://www.unicode.org/versions/Unicode4.0.1>) and by Unicode 4.1.0
169 (<http://www.unicode.org/versions/Unicode4.1.0>), March, 2005.
- 170 **[UUID]** Open Systems Interconnection – *Remote Procedure Call*, ISO/IEC
171 11578:1996, <http://www.iso.org/>, August 2001.
- 172 **[XML]** T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau,
173 *Extensible Markup Language (XML) 1.0, Third Edition*, World Wide Web
174 Consortium, <http://www.w3.org/TR/REC-xml/>, February 2004.
- 175 **[XMLDSig]** D. Eastlake, J. Reagle, D. Solo et al., *XML-Signature Syntax and*
176 *Processing*, World Wide Web Consortium,
177 <http://www.w3.org/TR/xmlsig-core/>, February, 2002.

178	[XMLID]	J. Marsh, D. Veillard, N. Walsh, <i>xml:id Version 1.0</i> , World Wide Web Consortium, http://www.w3.org/TR/2005/REC-xml-id-20050909 , September 2005.
179		
180		
181	[XMLSchema]	H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, <i>XML Schema Part 1: Structures Second Edition</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlschema-1/ , October 2004.
182		
183		
184	[XMLSchema2]	P. Biron, A. Malhotra, <i>XML Schema Part 2: Datatypes Second Edition</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlschema-2/ , October 2004.
185		
186		
187	[XRIMetadata]	D. Reed, <i>Extensible Resource Identifier (XRI) Metadata V2.0</i> , http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf , March 2005.
188		
189		
190	[XRISyntax]	D. Reed, D. McAlpin, <i>Extensible Resource Identifier (XRI) Syntax V2.0</i> , http://docs.oasis-open.org/xri/xri/V2.0/xri-syntax-V2.0-cd-01.pdf , March 2005.
191		
192		

193 1.6 Non-Normative References

194	[XRIFAQ]	OASIS XRI Technical Committee, <i>XRI 2.0 FAQ</i> , http://www.oasis-open.org/committees/xri/faq.php , Work-In-Progress, March 2006.
195		
196	[XRIGuide]	[TODO]
197	[XRIReqs]	G. Wachob, D. Reed, M. Le Maitre, D. McAlpin, D. McPherson, <i>Extensible Resource Identifier (XRI) Requirements and Glossary v1.0</i> , http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc , June 2003.
198		
199		
200		
201		
202	[Yadis]	TODO

203 **2 Conformance**

204 This section specifies conformance and conformance claims.

205 **2.1 Conformance Targets**

206 The targets of this specification are:

- 207 1. XRI resolvers, which may be either local resolvers or proxy resolvers.
208 2. XRI authority servers.

209 Note that a single implementation may serve all three functions, i.e., an XRI authority server may
210 also incorporate a local resolver and function as a proxy resolver.

211 **2.2 XRI Resolvers**

212 **2.2.1 Local Resolvers**

213 [TODO]

214 **2.2.2 Proxy Resolvers**

215 [TODO]

216 **2.3 XRI Authority Servers**

217 [TODO]

218 3 Namespaces

219 3.1 XRI Namespaces for XRI Resolution

220 As defined in section 2.2.1.2 of [XRISyntax], the GCS symbol “\$” is reserved for specified
221 identifiers, i.e., those assigned and defined by XRI TC specifications, other OASIS specifications,
222 or other standards bodies. (See also [XRIMetadata].) This section specifies the \$ namespaces
223 reserved for XRI resolution.

224 3.1.1 XRIs Reserved for XRI Resolution

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

XRI (in URI-Normal Form)	Usage	See Section
xri://\$res	Namespace for XRI resolution service types	3.1.2
xri://\$xrds	Namespace for the generic XRDS (Extensible Resource Descriptor Sequence) schema (not versioned)	3.2
xri://\$xrd	Namespace for the XRD (Extensible Resource Descriptor) schema	3.2
xri://\$xrd*(\$v*2.0)	Version 2.0 of above (using an XRI version identifier as defined in [XRIMetadata])	3.2

227 Table 2: XRIs reserved for XRI resolution.

228 3.1.2 XRIs Assigned to XRI Resolution Service Types

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

XRI	Usage	See Section
xri://\$res*auth	Authority resolution service	7
xri://\$res*auth*(\$v*2.0)	Version 2.0 of above	7
xri://\$res*proxy	HTTP(S) proxy resolution service	9
xri://\$res*proxy*(\$v*2.0)	Version 2.0 of above	9

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

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

234 3.2 XML Namespaces for XRI Resolution

235 Throughout this document, the following XML namespace prefixes have the meanings defined in
236 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 3.1.1 of this document
xrd	xri://\$xrd*(\$v*2.0)	Section 3.1.1 of this document

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

238 3.3 Media Types for XRI Resolution

239 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as
 240 defined by [RFC2046], particularly in HTTP Accept headers as specified in [RFC2616]. Table 5
 241 specifies the media types used for XRI resolution. Note that these media types are only used as
 242 HTTP Accept headers in XRI authority resolution, not in proxy resolution, where they are passed
 243 as query parameters in an HTTP(S) URI. See section 9.

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 10	Section 5 of [RFC2483]

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

245 To provide full control of XRI resolution via an HTTP interface, the media types specified in Table
 246 5 accept the media type parameters defined in Table 6.

Parameter	Values	Applies to Media Type	Usage
https	true false	application/xrds+xml application/xrd+xml text/uri-list	Specifies use of HTTPS trusted resolution (section 8.1)
saml	true false	application/xrds+xml application/xrd+xml text/uri-list	Specifies use of SAML trusted resolution (section 8.2)
nodefault	type path mediatype	application/xrds+xml application/xrd+xml text/uri-list	Specifies whether a default match of a service endpoint selection element is allowed (section 10.3)
cid	true false	application/xrds+xml application/xrd+xml	Specifies whether canonical ID verification must be performed

Comment [DSR8]: QUESTION: can you provide more than one value?

		text/uri-list	(section 12)
refs	true false	application/xrds+xml application/xrd+xml text/uri-list	Specifies whether references should be followed during resolution (section 0)
sep	true false	application/xrds+xml application/xrd+xml	Specifies whether service endpoint selection should be performed (section 10)

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

248 See sections 5 - 10 for more about usage of these media types and parameters.

249

4 XRDS Documents

250
251
252
253
254

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

255

4.1 XRDS and XRD Namespaces

256
257
258
259
260

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

261
262
263
264
265

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

266
267
268
269

This namespace architecture enables the XRDS namespace to remain constant while the XRD namespace (and the namespaces of other XML elements that may be included in an XRDS document) may be versioned over time. See section 16.2 for more about versioning of the XRD schema.

270

4.2 XRD Elements and Attributes

271
272

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

273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298

```
<XRDS xmlns="xri://$xrds" ref="xri://(http%3A%2F%2Fexample.org)*foo">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*foo</Query>
    <Status code="100"/>
    <Expires>2005-05-30T09:30:10Z</Expires>
    <ProviderID>urn:uuid:c9f812f3-6544-4e3c-874e-
d3ae79f4ef7b</ProviderID>
    <LocalID>*baz</LocalID>
    <CanonicalID>xri://(http%3A%2F%2Fexample.org)!1234!5678
</CanonicalID>
    <Ref>xri://!14a76!c2f7!9033</Ref>
    <Service>
      <ProviderID>xri://!1000!1234.5678</ProviderID>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <MediaType>application/xrds+xml</MediaType>
      <URI priority="10">http://resolve.example.com</URI>
      <URI priority="15">http://resolve2.example.com</URI>
      <URI>https://resolve.example.com</URI>
    </Service>
    <Service>
      <ProviderID>xri://!1000!1234.5678</ProviderID>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <MediaType>application/xrds+xml;https=true</MediaType>
      <URI>https://resolve.example.com</URI>
    </Service>
  </Service>
</XRD>
</XRDS>
```

```

299     <Type match="null" />
300     <Path select="true">media/pictures</Path>
301     <MediaType select="true">image/jpeg</MediaType>
302     <URI append="path" >http://pictures.example.com</URI>
303   </Service>
304   <Service>
305     <Type match="null" />
306     <Path select="true">media/videos</Path>
307     <MediaType select="true">video/mpeg</MediaType>
308     <URI append="path" >http://videos.example.com</URI>
309   </Service>
310   <Service>
311     <ProviderID> xri://!!11000!1234.5678</ProviderID>
312     <Type match="null" />
313     <Path match="default" />
314     <URI>http://example.com/local</URI>
315   </Service>
316   <Service>
317     <Type>http://example.com/some/service/v3.1</Type>
318     <URI>http://example.com/some/service/endpoint</URI>
319   </Service>
320 </XRD>
321 </XRDS>

```

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

325 4.2.1 Management Elements

326 The first set of elements are used to manage XRDs, particularly from the perspective of caching
327 and error handling.

328 **xrd:XRD**

329 Container element for all other XRD elements. Includes an OPTIONAL `xml:id` attribute
330 of type `xs:ID`. This attribute is REQUIRED in trusted resolution to uniquely identify this
331 element within the containing `xrd:XRDS` document. It also includes an OPTIONAL
332 `xrd:idref` attribute of type `xs:idref`. This attribute is REQUIRED in trusted resolution
333 when an XRD element in a nested `xrd:XRDS` document must reference a previously
334 included XRD instance. See sections 4.3 and 13.1. Lastly, it includes an `xrd:version`
335 attribute that is OPTIONAL for uses outside of XRI resolution but REQUIRED for XRI
336 resolution as defined in section 4.3.2

337 **xrd:XRD/xrd:Query**

338 0 or 1 per `xrd:XRD` element. Expresses the XRI, IRI, or URI reference in URI-normal
339 form whose resolution results in this `xrd:XRD` element. For XRI authority resolution, this
340 must be a qualified subsegment of the authority component of the query XRI.

341 **xrd:XRD/xrd:Status**

342 0 or 1 per `xrd:XRD` element. Contains a REQUIRED attribute `xrd:code` of type
343 `xs:int` that provides a numeric status code. The contents of the element are a human-
344 readable message string describing the status of the response. For XRI resolution,
345 values of the Status element and `xrd:code` attribute are defined in section 14.

346 **xrd:XRD/xrd:Expires**

347 0 or 1 per `xrd:XRD` element. The date/time, in the form of `xs:dateTime`, after which
348 this XRD cannot be relied upon. To promote interoperability, this date/time value
349 SHOULD use the UTC "Z" time zone and SHOULD NOT use fractional seconds. A
350 resolver using this XRD MUST NOT use the XRD after the time stated here. A resolver
351 MAY discard this XRD before the time indicated in this result. If the HTTP transport
352 caching semantics specify an expiry time earlier than the time expressed in this attribute,
353 then a resolver MUST NOT use this XRD after the expiry time declared in the HTTP
354 headers per section 13.2 of [RFC2616]. See section 15.2.1.

355 4.2.2 Authority Trust Elements

356 The second set of elements are for applications where trust must be established in the authority
357 providing the XRD. These elements are OPTIONAL for generic authority resolution (section 7),
358 but REQUIRED for specific types of trusted authority resolution (section 8).

359 **xrd:XRD/xrd:ProviderID**

360 0 or 1 per `xrd:XRD`. A unique identifier of type `xs:anyURI` for the authority producing
361 this XRD. The value of this element MUST be a persistent identifier. There MUST be
362 negligible probability that the value of this element will be assigned as an identifier to any
363 other authority. For purposes of CanonicalID verification (section 12), it is
364 RECOMMENDED to use a fully persistent XRI as defined in [XRISyntax]. If a URN
365 [RFC2141] or other persistent identifier is used, it is RECOMMENDED to express it as an
366 XRI cross-reference as defined in [XRISyntax]. Note that for XRI authority resolution, the
367 authority identified by this element is the *parent* authority (the provider of the current
368 XRD), not the *child* authority (the target of the current XRD). The latter is identified by the
369 `xrd:XRD/xrd:Service/xrd:ProviderID` element inside a resolution service
370 endpoint (see below).

371 **xrd:XRD/saml:Assertion**

372 0 or 1 per `xrd:XRD`. A SAML assertion from the *parent* authority (the provider of the
373 current XRD) that asserts that the information contained in the current XRD is
374 authoritative. Because the assertion is digitally signed and the digital signature
375 encompasses the containing `xrd:XRD` element, it also provides a mechanism for the
376 recipient to detect unauthorized changes since the time the XRD was published.

377 Note that while a `saml:Issuer` element is required within a `saml:Assertion` element,
378 this specification makes no requirement as to the value of the `saml:Issuer` element. It
379 is up to the XRI community resolution root to place restrictions, if any, on the
380 `saml:Issuer` element. A suitable approach is to use an XRI in URI-normal form that
381 identifies the community root authority. See section 7.1.3.

Comment [DSR9]: This section was completely rewritten to correspond to the new Synonym section (section 10).

382 4.2.3 Authority Synonym Elements

383 In the context of XRI architecture, two identifiers are *synonyms* if they are not character-for-
384 character equivalent but identify the same target resource. The normative rules for use of
385 synonyms in XRI resolution are specified in section 11. All synonym elements except
386 `<CanonicalID>` allow multiple instances, so all but `<CanonicalID>` accept the optional global
387 `xrd:priority` attribute (see section 4.3.3).

388 `xrd:XRD/xrd:LocalID`

389 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be a relative identifier. MUST
390 be assigned by the same parent authority providing the current XRD. Expresses a
391 interchangeable synonym for the value of the `xrd:Query` element. MAY be used to
392 verify a GlobalID as defined in section 12.

393 `xrd:XRD/xrd:GlobalID`

394 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier.
395 MUST be assigned by the same parent authority providing the current XRD. Expresses
396 an absolute identifier for the child authority for which the parent authority is authoritative.
397 MAY be verified as defined in section 12.

398 `xrd:XRD/xrd:Ref`

399 0 or more per `xrd:XRD` element. MUST be an absolute identifier. MUST be assigned by
400 a different parent authority than the parent authority producing the current XRD.
401 Expresses a forward reference as defined in section 11.1. MAY be processed to locate
402 additional XRDs documents describing the child authority as defined in section 13.1.

403 `xrd:XRD/xrd:Backref`

404 0 or more per `xrd:XRD` element. MUST be an absolute identifier. MUST be assigned by
405 a different parent authority than the parent authority producing the current XRD.
406 Expresses a backward reference as defined in section 11.1. MAY be used for synonym
407 verification as defined in section 12.

408 `xrd:XRD/xrd:CanonicalID`

409 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier. MAY be
410 assigned either by the parent authority producing the current XRD or by a different parent
411 authority. Expresses the canonical identifier for the child authority, i.e., the preferred
412 synonym among all synonyms. May be verified as defined in section 12.

413 4.2.4 Service Endpoint Management Elements

414 The next set of elements are used to describe service endpoints—the set of network endpoints
415 advertised in an XRD for performing further resolution, obtaining further metadata, or interacting
416 directly with the described resource. Again, because there can be more than one instance of a
417 service endpoint that satisfies a service endpoint selection query, or more than one instance of a

418 service endpoint URI, these elements both have the global `xrd:priority` attribute (see section
419 4.3.3).

420 **xrd:XRD/xrd:Service**

421 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.
422 Referred to by the abbreviation *SEP*.

423 **xrd:XRD/xrd:Service/xrd:URI**

424 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:anyURI`. If present, it
425 indicates a transport-level URI for access to the capability described by the parent
426 Service element. For the service types defined for XRI resolution in section 3.1.2, this
427 URI MUST be an HTTP or HTTPS URI. Other services may use other transport
428 protocols. This element includes an `xrd:append` attribute that governs construction of
429 the final service endpoint URI. See section 10.7.

430 **4.2.5 Service Endpoint Synonym Elements**

Comment [DSR10]: Rewrote in ED03 to correspond to the new Synonym section 10.

431 These are similar to the authority synonym elements except they provide synonyms for the child
432 authority in the context of a specific service endpoint. The normative rules for use of synonyms in
433 XRI resolution are specified in section 11. These elements all have the optional global
434 `xrd:priority` attribute (see section 4.3.3).

435 **xrd:XRD/xrd:Service/xrd:LocalID**

436 0 or more per `xrd:XRD/xrd:Service` element. Identical to `xrd:XRD/xrd:LocalID`
437 above except the contents may be either a relative or an absolute identifier. MAY be used
438 to provide one or more identifiers by which the target resource SHOULD be identified in
439 the context of the containing service endpoint.

440 **xrd:XRD/ xrd:Service/xrd:Ref**

441 0 or more per `xrd:XRD/xrd:Service` element. MUST be an absolute identifier.
442 Identical to `xrd:XRD/xrd:Ref` except this identifier serves as a forward reference to
443 another XRDS document. MAY be processed to locate additional XRDS documents with
444 additional service endpoint metadata as defined in section 13.2.

445 **4.2.6 Service Endpoint Trust Elements**

446 Similar to the authority trust elements, these elements enable trust to be established in the
447 provider of the service endpoint. These elements are OPTIONAL for generic authority resolution
448 (section 7), but REQUIRED for SAML trusted authority resolution (section 8.2).

449 **xrd:XRD/xrd:Service/xrd:ProviderID**

450 0 or 1 per `xrd:XRD/xrd:Service` element. Identical to the
451 `xrd:XRD/xrd:ProviderID` above, except this identifies the provider of the *described*
452 *service endpoint* instead of the provider of the current XRD. In SAML trusted resolution,
453 when a resolution request is made to the authority at this service endpoint, the contents
454 of the `xrd:XRD/xrd:ProviderID` element in the response MUST match the content of
455 this element for correlation. See section 8.2.5. The same usage MAY apply to other
456 services not defined in this specification, however that is beyond the scope of this
457 specification. Authors of other specifications employing XRD service endpoints SHOULD
458 define the scope and usage of this element, particularly for trust verification.

459 **xrd:XRD/xrd:Service/ds:KeyInfo**

460 0 or 1 per `xrd:XRD/xrd:Service` element. Provides the digital signature metadata
461 necessary to validate an XRD provided as a resolution response by the described
462 authority. This element comprises the key distribution method for SAML trusted authority
463 resolution in the XRI resolution framework—see section 8.2.5. The same usage MAY
464 apply to other services not defined in this specification.

465 4.2.7 Service Endpoint Selection Elements

466 The final set of elements are used in XRI resolution to select service endpoints. They include two
467 global attributes used for this purpose: `xrd:match` and `xrd:select`. See sections 10.2 and
468 10.4.

469 `xrd:XRD/xrd:Service/xrd:Type`

470 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI`
471 that identifies the type of capability available at this service endpoint. See section 3.1.2
472 for the resolution service types defined in this specification. If a service endpoint does not
473 include at least one `xrd:Type` element, the service type is effectively described by the
474 type of URI specified in the `xrd:XRD/xrd:Service/xrd:URI` element, i.e., an HTTP
475 URI specifies an HTTP service.

476 `xrd:XRD/xrd:Service/xrd:MediaType`

477 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of
478 content available at this service endpoint. The value of this element MUST be of the form
479 of a media type defined in [RFC2046]. See section 3.3 for the media types used in XRI
480 resolution.

481 `xrd:XRD/xrd:Service/xrd:Path`

482 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string
483 value meeting the `xri-path` production defined in section 2.2.3 of [XRISyntax].

484 The XRD schema (Appendix A) allows other elements and attributes from other namespaces to
485 be added throughout. As described in section 16.1.1, these points of extensibility can be used to
486 deploy new XRI resolution schemes, new service description schemes, or other metadata about
487 the described resource.

488 4.3 XRD Attribute Processing Rules

489 4.3.1 ID Attribute

490 For uses such as XRI trusted resolution (section 8.2) that require unique identification of multiple
491 XRD elements within an XRDS document, the XRD element uses an optional `xml:id` attribute
492 as defined by the W3C XML ID specification [XMLID]. If present, the value of this element MUST
493 be unique for all elements in the containing XML document. Because an XRI resolver may need
494 to assemble multiple XRDs received from different authority resolution services into one XRDS
495 document, there MUST be negligible probability that the value of the `xrd:XRD/@xml:id`
496 attribute is not globally unique. For this reason the value of this attribute SHOULD be a UUID as
497 defined by [UUID] prefixed by a single underscore character "_" in order to make it a legal
498 *NCName* as required by [XMLID]. However the value of this attribute MAY be generated by any
499 algorithm that fulfills the same requirements of global uniqueness and *NCName* conformance.

500 Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their
501 XML document order MUST match the order in which they were resolved (see section 7.1.2).
502 Also, if reference processing requires the same XRD to be included in an XRDS document twice
503 (via a nested XRDS document), that XRD MUST reference the previous instance using the
504 `xrd:XRD/@xml:idref` attribute as defined in section 13.1.2.

505 4.3.2 Version Attribute

506 Unlike the XRDS element, which is not intended to be versioned, the `xrd:XRD` element has the
507 optional attribute `xrd:XRD/@xrd:version`. Use of this attribute is REQUIRED for XRI
508 resolution. The value of this attribute MUST be the exact numeric version value of the XRI
509 Resolution specification to which the containing XRD element conforms. See section 3.1.1.

510 For more about versioning of the XRI resolution protocol, see section 16.2.

511 4.3.3 Priority Attribute

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

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

- 520 3. The client SHOULD select the element instance with the lowest numeric value of the
521 priority attribute. For example, an element with priority attribute value of "10" should be
522 selected before an element with a priority attribute value of "11", and an element with
523 priority attribute value of "11" should be selected before an element with a priority
524 attribute value of "25". Zero is the highest priority attribute value. Null is the lowest priority
525 attribute value—it is the equivalent of a value of infinity. It is RECOMMENDED to use a
526 large finite value (100 or more) rather than a null value.
- 527 4. If an element has no priority attribute, its priority attribute value is considered to be null,
528 i.e., the lowest possible priority value. Rather than omitting a priority attribute, it is
529 RECOMMENDED that XRI authorities follow the standard practice in DNS and set the
530 default priority attribute value to "10".
- 531 5. If two or more instances of the same element type have identical priority attribute values
532 (including the null value), the client SHOULD select one of the instances at random. This
533 client SHOULD NOT simply choose the first instance that appears in XML document
534 order (this is important in order to support intentional load balancing).
- 535 6. An element selected according to these rules is referred to in this specification as "the
536 highest priority element". If this element is subsequently disqualified from the set of
537 qualified elements, the next element selected according to these rules is referred to as
538 "the next highest priority element". If an XRI resolution operation specifying selection of
539 the highest priority element fails, the resolver SHOULD attempt to select the next highest
540 priority element unless otherwise specified. This process SHOULD be continued for all
541 other instances of the qualified elements until success is achieved or all instances are
542 exhausted.

543 4.4 XRI and IRI Encoding Requirements

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

547 A further restriction applies to XRIs or IRIs used in XRI resolution because it relies on HTTP(S) as
548 a transport protocol. Therefore when an XRI or IRI is used as the value of an `xrd:Query`,
549 `xrd:LocalID`, `xrd:GlobalID`, `xrd:CanonicalID`, `xrd:XRD/xrd:Ref`, `xrd:Backref`,
550 `xrd:Type`, or `xrd:Path` element, it MUST be in URI-normal form as defined in section 2.3 of
551 [XRISyntax].

552 Note: XRIs composed entirely of valid URI characters and which do not use XRI parenthetical
553 cross-reference syntax do not require escaping in the transformation to URI-normal form.

554 However XRIs that use characters valid only in IRIs or that use XRI parenthetical cross-reference
555 syntax may require percent encoding in the transformation to URI-normal form as explained in
556 section 2.3 of [XRISyntax].

557

5 Discovering an XRDS Document from an HTTP(S) URI

558

559

A resource described by an XRDS document and potentially identified by one or more XRI's may also be identified with one or more HTTP(S) URIs. For backwards compatibility with HTTP(S) infrastructure, this section defines a protocol, originally specified by Yadis.org, for discovering an XRDS document starting with an HTTP(S) URI.

561

562

Comment [DSR11]: This section was new in WD11 and has been thoroughly vetted on the OpenID spec list. Changes new in ED03 are shown with change marks.

Comment [DSR12]: Do we need to reference?

563

5.1 Overview

564

The protocol has two options: using an HTTP HEAD request to obtain a header with XRDS document location information (section 5.2), or using an HTTP GET request with content negotiation (section 5.3). A service hosting an XRDS document discoverable through an HTTP(S) URI MUST support the GET option, and MAY support the HEAD option. A resolving agent seeking to discover an XRDS document from an HTTP(S) URI MAY use either option, but MUST attempt the GET option if the HEAD option fails.

565

566

567

568

569

570

5.2 Protocol Using an HTTP HEAD Request

571

Under this protocol, to discover an XRDS document from an HTTP(S) URI, the requesting agent MUST begin by issuing an HTTP(S) HEAD request. This request SHOULD include an Accept header specifying the content type `application/xrds+xml`.

572

573

574

The response MUST be HTTP(S) response-headers only, which MAY include one or both of the following:

575

576

1. An `X-XRDS-Location` response-header.

577

2. A content type response-header specifying the content type `application/xrds+xml`.

578

If the response includes the first option above, the value of the `X-XRDS-Location` response-header MUST be an HTTP(S) URI which gives the location of an XRDS document describing the target resource. The requesting agent MUST then request this document as specified in section 5.3.

579

580

581

582

If the response includes the second option above, the requesting agent MUST request the XRDS document from the original HTTP(S) URI as specified in section 5.3.

583

584

If the response includes both options above, the value of the `X-XRDS-Location` element in the HTTP(S) response-header MUST take precedence.

585

586

If response includes neither of the two options above, this protocol fails and the requesting agent MUST fall back to using the protocol specified in section 5.3.

587

588

In all cases the HTTP(S) status messages and error codes defined in [\[RFC2616\]](#) apply.

589

5.3 Protocol Using an HTTP GET Request

590

Under this protocol, to discover an XRDS document from an HTTP(S) URI, the requesting agent MUST begin by issuing an HTTP(S) GET request. This request SHOULD include an Accept header specifying the content type `application/xrds+xml`.

591

592

593

The response MUST be one of four options:

594

1. HTTP(S) response-headers only as defined in section 5.2.

595

2. HTTP(S) response-headers as defined in section 5.2 together with a document, which MAY be either document type specified in options 3 or 4 below.

596

597 3. A valid HTML document with a `<head>` element that includes a `<meta>` element with an
598 `http-equiv` attribute equal to `X-XRDS-Location`.

599 4. A valid XRDS document (content type `application/xrds+xml`).

600 If the response is only HTTP(S) response headers as defined in section 5.2, or if it includes any
601 document other than the two document types defined in the third and fourth above, the protocol
602 MUST proceed as defined in section 5.2, except that there is no fallback to this section if that
603 protocol fails.

604 If the response is only an HTML document as defined in the third option above, the value of the
605 `<meta>` element with an `http-equiv` attribute equal to `X-XRDS-Location` MUST be an
606 HTTP(S) URI which gives the location of an XRDS document describing the target resource. The
607 requesting agent MUST then request the XRDS document from this URI using an HTTP(S) GET.
608 This request SHOULD include an Accept header specifying the content type
609 `application/xrds+xml`.

610 If the response includes both an HTTP(S) response header and the HTML document defined in
611 the third option above, the value of the `X-XRDS-Location` element in the HTTP(S) response-
612 header MUST take precedence.

613 If the response includes an XRDS document as specified in the fourth option above, the protocol
614 has completed successfully.

615 In all cases the HTTP(S) status messages and error codes defined in **[RFC2616]** apply.

616 Note: If the server supports content negotiation, the response SHOULD include a `Vary:` header
617 to allow caches to properly interpret future requests. This header SHOULD be present even in the
618 case where the HTML page is returned (instead of an XRDS document).

619

6 XRI Resolution Inputs and Outputs

620
621
622
623
624
625

Logically, XRI resolution is a function invoked by an application to dereference an XRI into a descriptor of the target resource (or in some cases to a representation of the resource itself). This section defines the logical inputs and outputs of this function, however it does not specify a binding to a particular local resolver interface. A binding to an HTTP interface for XRI proxy resolvers is specified in section 9. For purposes of illustration, a non-normative, language-neutral API is suggested in Appendix C.

626

6.1 Inputs

627
628
629
630
631

Table 7 summarizes the logical input parameters to XRI resolution and whether they are applicable in the authority resolution phase (sections 7 and 8) or the service endpoint selection phase (section 10). In this specification, references to these parameters use the logical names in the first column. Local APIs MAY use different names for these parameters and MAY define additional parameters.

Logical Input Parameter Name	Type	Required/Optional	Default	Resolution Phase
QXRI (query XRI) including authority string, path string, and query string	xs:anyURI	Required	N/A	Authority resolution
Resolution Output Format	xs:string (media type)	Optional	Null	Authority resolution
Service Type	xs:anyURI	Optional	Null	Service endpoint selection
Service Media Type	xs:string (media type)	Optional	Null	Service endpoint selection

632

Table 7: Input parameters for XRI resolution.

633
634

The following general rules apply to all input parameters as well as to all XRD elements throughout this specification:

635
636
637
638

1. The presence of a input parameter or an XRD element with an empty value is treated as equivalent to the absence of that input parameter or XRD element.
2. From a programmatic standpoint, both conditions above are considered as equivalent to setting the value of that parameter or element to null.

639
640

The following sections specify additional validation and usage requirements that apply to each input parameter.

641 **6.1.1 QXRI (Authority String, Path String, and Query String)**

642 The QXRI (query XRI) is the only REQUIRED input parameter. Per [XRISyntax], a QXRI consists
 643 of three logical subparameters as defined in Table 8.

Logical Parameter Name	Type	Required/Optional	Value
Authority String	xs:string	Required	Contents of the authority segment of the QXRI, not including the XRI scheme name or leading double forward slashes ("/") or a terminating single forward slash ("/").
Path String	xs:string	Optional	Contents of the path component of the QXRI, not including the 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.

644 *Table 8: Subparameters of the QXRI input parameter.*

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

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

- 649 1. It **MUST** be a valid absolute XRI according to the ABNF defined in [XRISyntax]. To
 650 resolve a relative XRI reference, it must be converted into an absolute XRI using the
 651 procedure defined in section 2.4 of [XRISyntax].
- 652 2. For authority or proxy resolution as defined in this specification, the QXRI **MUST** be in
 653 URI-normal form as defined in section 2.3.1 of [XRISyntax]. A local resolver API **MAY**
 654 support the input of other XRI forms but **SHOULD** document the normal form(s) it
 655 supports and its normalization policies.
- 656 3. When a QXRI is included as part of an HXRI (section 9.2) for XRI proxy resolution, the
 657 QXRI **MUST** be normalized as specified in section 9.2, and all HXRI query parameters
 658 **MUST** follow the encoding rules specified in section 9.3.

659 **6.1.2 Resolution Output Format**

660 The Resolution Output Format is an OPTIONAL string that is used to specify:

- 661 • The media type for the resolution response.
- 662 • Whether generic or trusted resolution must be used by the resolver.
- 663 • Whether references should be followed during resolution.
- 664 • Whether CanonicalID verification should be performed during resolution.
- 665 • Whether service endpoint selection should be performed on the final XRD.
- 666 • Whether default matches should be ignored during service endpoint selection.

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

- 668 1. The value of Resolution Output Format MUST be one of the values specified in Table 5
669 and MAY include any of the media type parameters specified in Table 6.
- 670 2. If the value of the `https` media type parameter is `true`, the resolver MUST use the
671 HTTPS trusted authority resolution protocol specified in section 8.1 (or return an error
672 indicating this is not supported).
- 673 3. If the value of the `saml` media type parameter is `true`, the resolver MUST use the SAML
674 trusted authority resolution protocol specified in section 8.2 (or return an error indicating
675 this is not supported).
- 676 4. If the value of both the `https` and `saml` media type parameters is `true`, the resolver
677 MUST use the HTTPS+SAML trusted authority resolution protocol specified in section 8.3
678 (or return an error indicating this is not supported).
- 679 5. If the value of the `cid` media type parameter is `true`, the resolver MUST perform
680 CanonicalID verification as specified in section 12.
- 681 6. If the value of the `cid` media type parameter is `false` or null, or if the parameter is
682 absent, the resolver MUST NOT perform CanonicalID verification.
- 683 7. If the value of the `refs` media type parameter is `true` or null, or if the parameter is
684 absent, the resolver MUST perform reference processing as defined in section 0 if it is
685 necessary to complete resolution (or return an error indicating this is not supported).
- 686 8. If the value of the `refs` media type parameter is `false`, the resolver MUST NOT
687 perform reference processing during resolution.
- 688 9. If the value of the `sep` media type parameter is `true`, the resolver MUST perform service
689 endpoint selection on the final XRD (or return an error indicating this is not supported).
- 690 10. If the value of the `sep` media type parameter is `false` or null, or if the parameter is
691 absent, the resolver MUST NOT perform service endpoint selection on the final XRD
692 unless it is required to produce a URI List or HTTP(S) redirect. See section 6.2.
- 693 11. If the value of the `ndefault` media type parameter is `type`, `path`, or `mediatype`,
694 the resolver MUST ignore default service endpoint selection element matches as
695 specified in section 10.3.2.

696 Future versions of this specification, or other specifications for XRI resolution, MAY use other
697 values for Resolution Output Format or its media type parameters.

698 6.1.3 Service Type

699 The Service Type is an OPTIONAL value of type `xs:anyURI` used to request a specific type of
700 service in the service endpoint selection phase (section 9). The value of this parameter MUST be
701 a valid absolute XRI, IRI, or URI in URI-normal form as defined by **[XRISyntax]**. (Note that URI-
702 normal form is specified so that this parameter may be passed to a proxy resolver in a QXRI
703 query parameter as defined in section 9.) The Service Type values defined for XRI resolution
704 services are specified in section 3.1.2.

705 6.1.4 Service Media Type

706 The Service Media Type is an OPTIONAL string used to request a specific media type in the
707 service endpoint selection phase (section 9). The value of this parameter MUST be a valid media
708 type as defined by **[RFC2046]**. The Service Media Type values defined for XRI resolution
709 services are specified in section 3.3.

710 6.2 Outputs

711 Table 9 summarizes the logical outputs of XRI resolution.

Logical Output Format Name	Media Type Value (when requesting XRI authority resolution only)	Media Type Value (when 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>

712 Table 9: Outputs of XRI resolution.

713 The following sections provide additional construction and validation requirements.

714 6.2.1 XRDS Document

715 If the value of the Resolution Output Format parameter is `application/xrds+xml`, the
716 following rules apply.

- 717 1. The output MUST be a valid XRDS document according to the schema defined in
718 Appendix A. Also, any nested XRDS documents included as a result of reference
719 processing (section 12) must also be valid.
- 720 2. Each of the contained XRD elements must be a valid XRD document according to the
721 schema defined in Appendix A.
- 722 3. The XRD elements MUST conform to the additional requirements in section 4.
- 723 4. If the value of the `saml` parameter of the Resolution Output Format is `true`, the XRD
724 element MUST further conform to the additional requirements in section 8.2.
- 725 5. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST
726 be performed as defined in section 10, even if the values of all three service endpoint
727 selection input parameters (Service Type, Service Media Type, and Path String) are null.
728 This ensures that a resolver will perform reference processing (section 12) to locate a
729 requested service endpoint (or a default service endpoint) if necessary. IMPORTANT:
730 Regardless of whether reference processing is performed or not, no filtering of the final
731 XRD is performed when returning an XRDS document. Filtering is only performed when
732 the requested Resolution Output Format is an XRD document – see the next section.
- 733 6. If reference processing is necessary during the authority resolution or service endpoint
734 selection process, it MUST result in a nested XRDS document as defined in section
735 13.1.2. Again, no filtering of the final XRD is performed when returning an XRDS
736 document.
- 737 7. If the output is an error, this error MUST be returned using the `xrd:Status` element of
738 the final XRD in the XRDS document as defined in section 14.

739 6.2.2 XRD Document

740 If the value of the Resolution Output Format parameter is `application/xrd+xml`, the following
741 rules apply.

- 742 1. The output MUST be a valid XRD document according to the schema defined in
743 Appendix A.

- 744 2. The XRD elements MUST conform to the additional requirements in section 4.
- 745 3. [TODO – add processing instructions for the case where the `saml` media type parameter
- 746 is `true`.]
- 747 4. If the value of the `sep` media type parameter is `false` or null, or if this parameter is
- 748 absent, the XRD MUST be the final XRD in the XRDS document produced as a result of
- 749 authority resolution. Service endpoint selection or any other filtering of the XRD
- 750 document MUST NOT be performed.
- 751 5. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST
- 752 be performed as defined in section 10, even if the values of all three service endpoint
- 753 selection input parameters (Service Type, Service Media Type, and Path String) are null.
- 754 6. If service endpoint selection is performed, the only `xrd:Service` elements in the XRD
- 755 document MUST be those selected according to the rules specified in section 10. If no
- 756 service endpoints were selected by those rules, no `xrd:Service` elements will be
- 757 present. In addition, all elements in the XRD document that are subject to the global
- 758 `xrd:priority` attribute (even if the attribute is absent or null) MUST be returned in
- 759 order of highest to lowest priority as defined in section 4.3.3. Any other filtering of the
- 760 XRD document MUST NOT be performed. Note that this means that if the XRD
- 761 document includes a SAML signature element as defined in section 8.2, this element is
- 762 still returned in the XRD document even though it may not be able to be verified by a
- 763 consuming application.
- 764 7. If the output is an error, this error MUST be returned using the `xrd:Status` element as
- 765 defined in section 14.

766 6.2.3 URI List

767 If the value of the Resolution Output Format parameter is `text/uri-list`, the following rules

768 apply.

- 769 1. For this output, service endpoint selection is REQUIRED.
- 770 2. If authority resolution and service endpoint selection are both successful, the output
- 771 MUST be a valid URI List as defined by section 5 of [RFC2483].
- 772 3. If, after applying the service endpoint selection rules, more than one service endpoint is
- 773 selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as
- 774 defined in section 4.3.3.
- 775 4. If the final selected `xrd:XRD/xrd:Service` element does not contain an
- 776 `xrd:XRD/xrd:Service/xrd:URI` element but does contain at least one
- 777 `xrd:XRD/xrd:Service/xrd:Ref` element, service ref processing MUST be
- 778 performed as described in section 13.2.
- 779 5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)
- 780 MUST be constructed as defined in section 10.6.
- 781 6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`
- 782 elements within the selected `xrd:Service` element as defined in section 4.3.3. When
- 783 two or more of the source `xrd:URI` elements have equal priority, their constructed URIs
- 784 SHOULD be returned in random order. Any other filtering of the URI list MUST NOT be
- 785 performed.
- 786 7. If the output is an error, it MUST be returned with the content type `text/plain` as
- 787 defined in section 14.

788 6.2.4 HTTP(S) Redirect

789 In XRI proxy resolution, the Resolution Output Format parameter may be null. In this case the

790 output of a proxy resolver is an HTTP(S) redirect as defined in section 9.6.

791

7 Generic Authority Resolution

792
793
794

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

795
796
797
798
799

XRI authorities and IRI authorities have different syntactic structures, partially due to the higher level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved to XRDS documents one subsegment at a time as specified in section 7.1. IRI authorities, since they are based on DNS names or IP addresses, are resolved into an XRDS document through a special HTTP(S) request using the entire IRI authority segment as specified in section 7.2.

800

7.1 XRI Authority Resolution

801

7.1.1 Service Type and Service Media Type

802

The protocol defined in this section is identified by the values in Table 10.

Service Type	Service Media Type	Media Type Parameters
xri://\$res*auth*(\$v*2.0)	application/xrds+xml	Not applicable (see important note below)

803

Table 10: Service Type and Service Media Type values for generic authority resolution.

804
805

A generic authority resolution service endpoint advertised in an XRDS document MUST use the Service Type identifier and MAY use the Service Media Type identifier defined in Table 10.

806
807
808
809
810
811

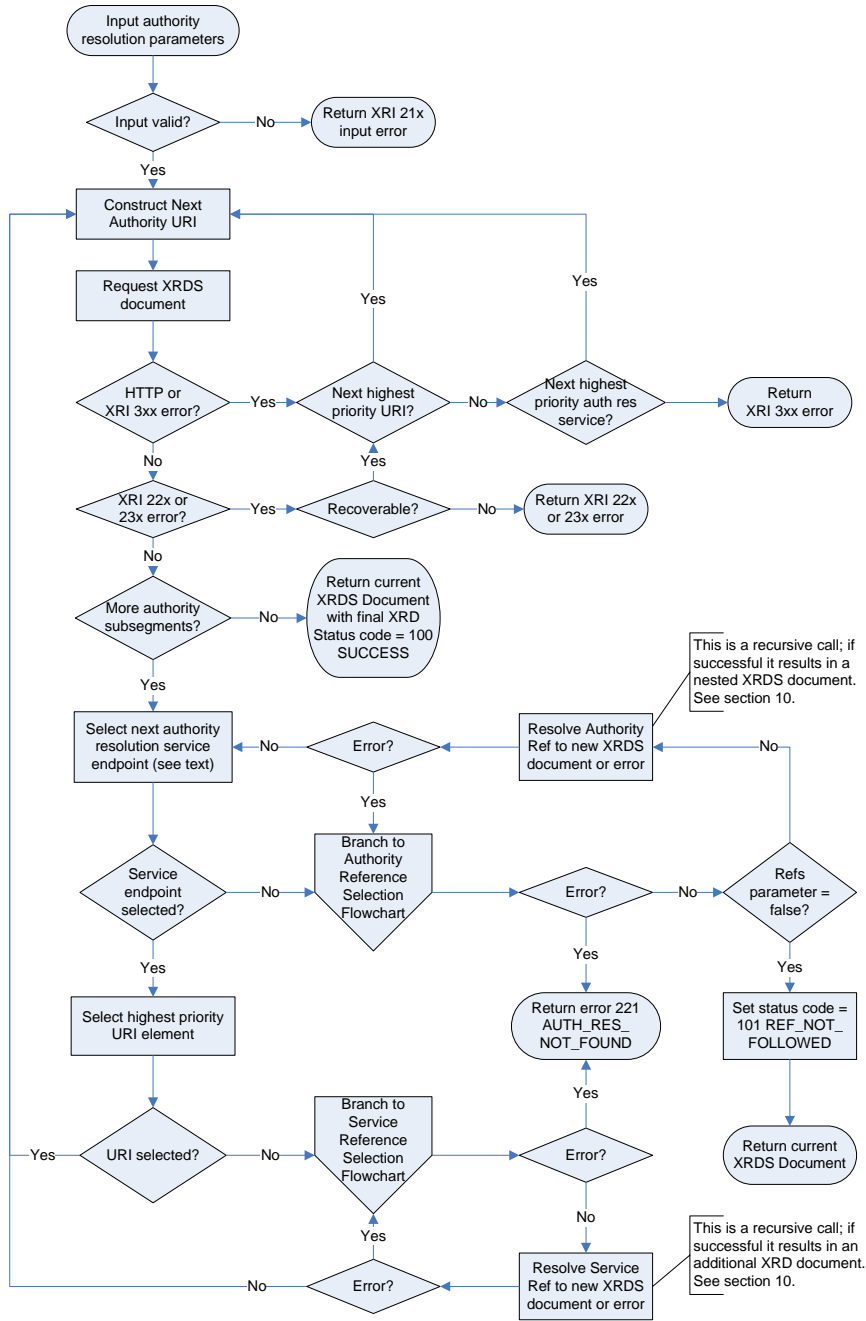
IMPORTANT NOTE FOR BACKWARDS COMPATABILITY: Earlier drafts of this specification used a media type parameter called `trust`. This has been deprecated in favor of new parameters for each trusted resolution option, i.e., `https="true"` and `saml="true"`. However implementations SHOULD consider the following values equivalent both for the purpose of service endpoint selection within XRDS documents and as HTTP(S) Accept header values in XRI authority resolution requests:

812
813
814
815
816
817

```
application/xrds+xml
application/xrds+xml;trust=none
application/xrds+xml;https=false
application/xrds+xml;saml=false
application/xrds+xml;https=false;saml=false
application/xrds+xml;saml=false;https=false
```

818 **7.1.2 Protocol**

819 Figure 2 (non-normative) shows the overall logical flow of generic authority resolution.



820

821 *Figure 2: Authority resolution flowchart*

822 Following are the normative requirements for behavior of an XRI resolver and an XRI resolution
823 service when performing generic XRI authority resolution:

- 824 1. The resolver MUST be preconfigured with the XRDS document describing the community
825 root authority for the XRI to be resolved as defined in section 7.1.3, or have another
826 equivalent means of obtaining this metadata. The resolver MAY obtain this using a self-
827 describing XRDS document as defined in section 7.1.4.
- 828 2. Resolution of each subsegment in the Authority String after the community root
829 subsegment MUST proceed in subsegment order (left-to-right) using fully qualified
830 subsegment values as defined in section 7.1.5. If the final output is an XRDS document,
831 this document MUST contain an ordered list of `xrd:XRDS` elements—one for each
832 authority subsegment successfully resolved by the resolver client. This list MUST appear
833 in the same order as the corresponding subsegments in the Authority String. In addition,
834 any authority references followed MUST be represented by nested `xrd:XRDS`
835 documents as defined in section 13.1.2 and any service references followed MUST be
836 represented by additional `xrd:XRDS` elements as defined in section 13.2.2.
- 837 3. Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as
838 defined in section 7.1.6.
- 839 4. The resolver MAY request that a recursing authority resolution service perform resolution
840 of multiple subsegments as defined in section 7.1.7.
- 841 5. For each iteration of the authority resolution process, the next authority resolution service
842 endpoint MUST be selected as specified in section 7.1.8 and an HTTP(S) URI (called the
843 Next Authority URI) MUST be constructed as specified in section 7.1.8.
- 844 6. Each resolution request MUST be an HTTP(S) GET to the Next Authority URI and MUST
845 contain an Accept header with the media type identifier defined in Table 10. Note that in
846 XRI authority resolution, this Accept header is NOT interpreted as an XRI resolution input
847 parameter, but simply as a media type. This differs from XRI proxy resolution, where the
848 Accept header MAY be used to specify the Service Media Type resolution parameter.
849 See section 9.4.
- 850 7. The ultimate HTTP(S) response from an authority resolution service to a successful
851 resolution request MUST contain either: a) a 2XX response with a valid XRDS document
852 containing an XRD element for each authority subsegment resolved, or b) a 304
853 response signifying that the cached version on the resolver is still valid (depending on the
854 client's HTTP(S) request). There is no restriction on intermediate redirects (i.e., 3XX
855 result codes) or other result codes (e.g., a 100 HTTP response) that eventually result in a
856 2XX or 304 response through normal operation of [RFC2616].
- 857 8. The HTTP(S) response from an authority resolution service MUST return the media type
858 requested by the resolver. The response SHOULD NOT include any media type
859 parameters supplied by the resolver in the request. If the resolver receives such
860 parameters in the response, the resolver MUST ignore them and do its own independent
861 verification that the response fulfills the requested parameters.
- 862 9. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in
863 the resolution process and the resolver SHOULD return the appropriate error code and
864 context message as specified in section 14. In recursing resolution, such an error MUST
865 be returned by the recursing authority resolution service to the resolver as specified in
866 section 14.3.
- 867 10. If an XRD does not include the next requested authority service endpoint but includes
868 one or more `xrd:Ref` elements, the resolver MUST perform reference processing as
869 defined in section 12 unless the `refs` media type parameter defined in Table 6 is set to
870 `false`. If the `refs` media type parameter is set to `false` and the XRD contains at least
871 one `xrd:Ref` element that could be followed, the resolver MUST return a response with
872 a status code of 101 `REF_NOT_FOLLOWED`. (Note that such reference processing, if

873 successful, will result in a separate nested XRDS document describing the resolved
874 reference as defined in section 13.1.2.)

875 11. A successful response that does not include the next required authority service endpoint
876 in the XRD and does not include any `xrd:Ref` elements MUST return an error with a
877 status code of 221 `AUTH_RES_NOT_FOUND` regardless of the value of the `refs` media
878 type parameter.

879 12. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section
880 15. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent
881 possible to maintain the efficiency and scalability of the HTTP-based resolution system.
882 The recommended use of HTTP caching headers is described in more detail in section
883 15.2.1.

884 7.1.3 Community Root Authorities

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

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

895 The community root authority SHOULD publish a self-describing XRDS document as defined in
896 section 7.1.4. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as
897 the community's root authority resolution service endpoints. This community root XRDS
898 document, or its location, must be known *a priori* and is part of the configuration of an XRI
899 resolver, similar to the specification of root DNS servers for a DNS resolver. Note that is not
900 strictly necessary to publish this information in an XRDS document—it may be supplied in any
901 format that enables configuration of the XRI resolvers in the community. However publishing a
902 self-describing XRDS document at a known location simplifies this process and enables dynamic
903 configuration of community resolvers.

904 It is also a recommended best practice for a community root XRDS document to contain:

- 905 • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- 906 • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML
907 trusted resolution is supported.
- 908 • Both of the above if HTTPS+SAML trusted resolution is supported.
- 909 • The service endpoints and supported media types of the community's XRI proxy resolver(s) if
910 proxy resolution is supported.

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

Comment [DSR13]: OPEN ISSUE:
remains open for discussion.

914 7.1.4 Self-Describing XRDS Documents

915 An identifier authority MAY publish a self-describing XRDS document, i.e., one produced by the
916 same identifier authority that it describes. A resolver MAY request a self-describing XRDS
917 document from a target identifier authority using either of two methods:

- 918 1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution
919 service endpoint, it may use the resolution protocol specified in section 5 to request an

920 XRDS document directly from this HTTP(S) URI(s). This HTTP(S) URI may be known a
921 priori (as is often the case with community root authorities, above), or it may be
922 discovered from other identifier authorities via the resolution protocols defined in this
923 specification.

924 2. If the resolver knows: a) an XRI of the target authority as a community root authority, and
925 b) the location of a proxy resolver configured for this community root authority, it may use
926 the proxy resolution protocol specified in section 9 to query the proxy resolver for the
927 community root authority XRI. This query MUST include only a single subsegment
928 identifying the community root authority and MUST NOT include any additional
929 subsegments.

930 An example of the first method, if a identifier authority had an authority resolution service
931 endpoint at `http://example.com/auth-res-service/`, would be to issue an HTTP(S) GET
932 request to that URI with an Accept header specifying the content type
933 `application/xrds+xml`. See section 5.3 for more details.

934 An example of the second method, if a identifier authority had the community root authority
935 identifier `xri://(example)` and was registered with the XRI proxy resolver
936 `http://xri.example.com/`, would be to issue an HTTP(S) GET request to the following URI:
937 `http://xri.example.com/(example)?_xrd_r=application/xrds+xml`

938 See section 9 for more details.

939 Note that a proxy resolver may use the first method to publish its own self-describing XRDS
940 document at the HTTP(S) URI(s) for its proxy resolution service.

941 **IMPORTANT:** A self-describing XRDS document MUST only be issued by an identifier authority
942 when describing itself. It MUST NOT be included when the identifier authority is describing a
943 different identifier authority. In the latter case the self-describing XRDS document for the
944 community root authority is implicit. It MAY be explicitly requested by the resolver if needed for
945 dynamic configuration of the resolver or trust verification of other XRI resolution chains.

946 7.1.5 Qualified Subsegments

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

951 If the first subsegment of an XRI authority is a GCS character and the following subsegment does
952 not begin with a “*” (indicating a reassignable subsegment) or a “!” (indicating a persistent
953 subsegment), then a “*” is implied and MUST be added when constructing the qualified
954 subsegment as specified in section 7.1.8. Table 11 and Table 12 illustrate the differences
955 between parsing a reassignable subsegment following a GCS character and parsing a cross-
956 reference, respectively.

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

958 *Table 11: Parsing the first subsegment of an XRI that begins with a global context symbol.*

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

959 *Table 12: Parsing the first subsegment of an XRI that begins with a cross-reference.*960 **7.1.6 Cross-References**

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

965 Table 13 provides several examples of resolving cross-references. In these examples,
 966 subsegment "lb" resolves to a Next Authority Service Endpoint URI of "http://example.com/xri-
 967 authority/" and recursing authority resolution is not being requested.

968

Cross-reference type	Example XRI	Next Authority URI after resolving "xri://@!a!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)

969 *Table 13: Examples of the Next Authority URIs constructed using different types of cross-references.*970 **7.1.7 Recursing Authority Resolution**

971 If an authority resolution service offers recursing resolution, an XRI resolver may request
 972 resolution of multiple authority subsegments in one transaction. If a resolver makes such a
 973 request, the responding authority resolution service MAY perform the additional recursing
 974 resolution steps requested. In this case the recursing authority resolution service acts as a
 975 resolver to the other authority resolution service endpoints that need to be queried. Alternatively,
 976 the recursing authority resolution service may retrieve XRDs from its local cache until it reaches a
 977 subsegment whose XRD is not locally cached, or it may simply recurse only as far as it is

978 authoritative. If an authority resolution service performs any recursing resolution, it MUST return
979 an ordered list of `xrd:XRDS` elements (and nested `xrd:XRDS` elements if references are
980 followed) in an `xrd:XRDS` document for all subsegments resolved as defined in section 7.1.2.
981 The recursing authority resolution service MAY resolve fewer subsegments than requested by the
982 resolver. The recursing authority resolution service is under no obligation to resolve more than
983 the first subsegment (for which it is, by definition, authoritative).
984 If the recursing authority resolution service does not resolve the entire set of subsegments
985 requested, the resolver is responsible for continuing the authority resolution process itself. At any
986 stage, however, the resolver MAY request that the next authority resolution service recursively
987 resolve any remaining subsegments.

988 7.1.8 Construction of the Next Authority URI

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

993 The process for selecting the next authority resolution service endpoint from the current XRD is
994 defined in section 10. For generic authority resolution, this selection process MUST use the
995 parameters specified in Table 10. For trusted authority resolution, this selection process MUST
996 use the parameters specified in Table 14, Table 15, or Table 16. In all cases, an explicit match on
997 the `xrd:XRDS/Service/xrd:Type` element is REQUIRED, so during authority resolution, a
998 resolver MUST set the `nodefault` parameter to a value of `nodefault=type` in order to
999 override selection of a default service endpoint as specified in section 10.3.2.

1000 From the output of the service endpoint selection process, the resolver MUST select the highest
1001 priority URI of the highest priority authority resolution service endpoint. Next, the resolver MUST
1002 apply the service endpoint URI construction algorithm based the value of the `append` attribute as
1003 defined in section 10.7.

1004 This fully constructed URI is called the *Next Authority Service Endpoint URI*. If this URI does not
1005 end with a forward slash (“/”), one MUST be appended before proceeding.

1006 The second string is called the *Next Authority String* and it consists of either:

- 1007 • The next fully qualified subsegment to be resolved (see section 7.1.5), or
- 1008 • In the case of recursing resolution, the next fully qualified subsegment to be resolved plus
1009 any additional subsegments for which recursing resolution is requested (see section 7.1.6).

1010 The final step is to append the Next Authority String to the path component of the Next Authority
1011 Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

1012 Construction of the Next Authority URI is more formally described in this pseudocode for
1013 resolving a “next-auth-string” via a “next-auth-sep-uri”:

```
1014 if (path portion of next-auth-sep-uri does not end in "/"):
1015     append "/" to path portion of next-auth-sep-uri
1016
1017 if (next-auth-string is not preceded with "*" or "!" delimiter):
1018     prepend "*" to next-auth-string
1019
1020 append uri-escape(next-auth-string) to path of next-auth-sep-uri
```

1021 7.2 IRI Authority Resolution

1022 From the standpoint of generic authority resolution, an IRI authority segment represents either a
1023 DNS name or an IP address at which an XRDS document describing the authority may be
1024 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET
1025 request to a URI constructed from the IRI authority segment. The resulting XRDS document can
1026 then be consumed in the same manner as one obtained using XRI authority resolution.

1027 While the use of IRI authorities provides backwards compatibility with the large installed base of
1028 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of
1029 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities
1030 are not recommended for new deployments of XRI identifiers.

1031 This section defines IRI authority resolution as a simple extension to the XRI authority resolution
1032 protocol defined in the preceding section.

1033 7.2.1 Service Type and Media Type

1034 Because IRI authority resolution takes place at a level “below” XRI authority resolution, it cannot
1035 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority
1036 resolution uses the same media type as generic XRI authority resolution.

1037 7.2.2 Protocol

1038 Following are the normative requirements for IRI authority resolution that differ from generic XRI
1039 authority resolution:

1040 1. The next authority URI is constructed by extracting the entire IRI authority segment and
1041 prepending the string “http://”. See the exception in section 7.2.3.

1042 2. The HTTP GET request MUST include an HTTP Accept header containing only the
1043 following:

1044 `Accept: application/xrds+xml`

1045 3. The HTTP GET request MUST have a Host: header (as defined in section 14.23 of
1046 **[RFC2616]**) containing the value of the IRI authority segment.

1047 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document
1048 containing the XRD describing that authority.

1049 5. The responding server MUST use the value of the Host header to populate the
1050 `xrd:XRD/xrd:Query` element in the resulting XRD. For example:

1051 `Host: example.com`

1052 Note that because IRI authority resolution is required to process the entire IRI authority segment
1053 in a single step, recursing authority resolution does not apply.

1054 7.2.3 Optional Use of HTTPS

1055 Section 8 of this specification defines trusted resolution only for XRI authorities. Trusted
1056 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to
1057 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY
1058 use HTTPS as the access protocol for retrieving the authority’s XRD. If the resolver is satisfied,
1059 via transport level security mechanisms, that the response is from the expected IRI authority, the
1060 resolver may consider this an HTTPS trusted resolution response as defined in section 8.1.

1061 8 Trusted Authority Resolution

1062 This section defines three options for performing trusted XRI authority resolution as an extension
1063 of the generic XRI authority resolution service defined in section 7.1—one using HTTPS, one
1064 using SAML assertions, and one using both.

1065 8.1 HTTPS

1066 This option for trusted authority resolution is a very simple addition to generic authority resolution
1067 in which all communication with authority resolution service endpoints is carried out over HTTPS.
1068 This provides transport-level security and server authentication, however it does not provide
1069 message-level security or a means for a responder to provide different responses for different
1070 requestors.

1071 8.1.1 Service Type and Service Media Type

1072 The protocol defined in this section is identified by the values in Table 14.

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

1073 *Table 14: Service Type and Service Media Type values for HTTPS trusted authority resolution.*

1074 An HTTPS trusted resolution service endpoint advertised in an XRDS document MUST use the
1075 Service Type identifier and Service Media Type identifier (including the `https=true` parameter)
1076 defined in Table 14. In addition, the identifier authority MUST use an HTTPS URI as the value of
1077 the `xrd:URI` element(s) for this service endpoint.

1078 8.1.2 Protocol

1079 Following are the normative requirements for HTTPS trusted authority resolution that differ from
1080 generic XRI authority resolution (section 7.1):

- 1081 1. All authority resolution service endpoints MUST be selected using the values defined in
1082 Table 14.
- 1083 2. All authority resolution requests including the starting request to a community root
1084 authority MUST use the HTTPS protocol as defined in [RFC2818]. A successful HTTPS
1085 response MUST be received from each authority in the resolution chain or the resolver
1086 MUST output an error.
- 1087 3. All authority resolution requests MUST contain an HTTPS Accept header with the media
1088 type identifier defined in Table 14 (including the `https="true"` parameter).
- 1089 4. If the resolver finds that an authority in the resolution chain does not support HTTPS, the
1090 resolver MUST return a 23x error as defined in section 14.

1091 8.2 SAML

1092 In SAML trusted resolution, the resolver requests a content type of `application/xrds+xml;`
1093 `saml=true` and the authority resolution service responds with an XRDS document containing an
1094 XRD with an additional element—a digitally signed SAML [SAML] assertion that asserts the
1095 validity of the containing XRD. SAML trusted resolution provides message integrity but does not
1096 provide confidentiality. The latter may be achieved by combining it with HTTPS as defined in
1097 section 8.3. Message confidentiality may also be achieved with other security protocols used in

1098 conjunction with this specification. SAML trusted resolution also does not provide a means for an
1099 authority to provide different responses for different requestors; client authentication is explicitly
1100 out-of-scope for version 2.0 of XRI resolution.

1101 8.2.1 Service Type and Service Media Type

1102 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	saml=true

1103 *Table 15: Service Type and Service Media Type values for SAML trusted authority resolution.*

1104 A SAML trusted resolution service endpoint advertised in an XRD document MUST use the
1105 Service Type identifier and Service Media Type identifier defined in Table 15 (including the
1106 `saml=true` parameter). In addition, for transport security the identifier authority SHOULD offer at
1107 least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1108 8.2.2 Protocol

1109 8.2.2.1 Client Requirements

1110 For a resolver, trusted resolution is identical to the generic resolution protocol (section 7.1) with
1111 the addition of the following requirements:

- 1112 1. All authority resolution service endpoints MUST be selected using the values defined in
1113 Table 15. A resolver SHOULD NOT request SAML trusted resolution service from an
1114 authority unless the authority advertises a resolution service endpoint matching these
1115 values.
- 1116 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is
1117 recommended for confidentiality.
- 1118 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the
1119 media type identifier defined in Table 15 (including the `saml=true` parameter). This
1120 MUST be interpreted as the value of the Resolution Output Format input parameter.
1121 (Clients willing to accept either generic or trusted responses may use a combination of
1122 media type identifiers in the Accept header as described in section 14.1 of [RFC2616].
1123 Media type identifiers SHOULD be ordered according to the client's preference for the
1124 media type of the response. If a client performing generic authority resolution receives an
1125 XRD containing SAML elements, it is NOT REQUIRED to validate the signature or
1126 perform any processing of these elements.)
- 1127 4. A resolver MAY request recursing authority resolution of multiple subsegments as
1128 defined in section 8.2.3.
- 1129 5. The resolver MUST individually validate each XRD in the resolution chain according to
1130 the rules defined in section 8.2.4. When `xrd:XRD` elements come both from freshly-
1131 retrieved XRD documents and from a local cache, a resolver MUST ensure that these
1132 requirements are satisfied each time a resolution request is performed.

1133 8.2.2.2 Server Requirements

1134 For an authority resolution service, trusted resolution is identical to the generic resolution protocol
1135 (section 7.1) with the addition of the following requirements:

- 1136 1. The HTTP(S) response to a trusted resolution request MUST include a content type of
1137 "application/xrds+xml;trust=saml".

- 1138 2. The XRDs document returned by the resolution service MUST contain a
1139 `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid
1140 per the processing rules described by [SAML].
- 1141 3. The SAML assertion MUST contain a valid enveloped digital signature as defined by
1142 [XMLDSig] and as constrained by section 5.4 of [SAML].
- 1143 4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML
1144 assertion. Specifically, the signature MUST contain a single
1145 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference
1146 MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML
1147 assertion. The `URI` reference MUST NOT be empty and it MUST refer to the identifier
1148 contained in the `xrd:XRD/@xml:id` attribute.
- 1149 5. In [SAML], the digital signature enveloped by the SAML assertion may contain a
1150 `ds:KeyInfo` element. If this element is included, it MUST describe the key used to verify
1151 the digital signature element. Because the signing key is known in advance by the
1152 resolution client, the `ds:KeyInfo` element SHOULD be omitted from the digital
1153 signature.
- 1154 6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST
1155 match the XRI authority subsegment requested by the client.
- 1156 7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match
1157 the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD
1158 advertising availability of trusted resolution service from this authority as required in
1159 section 8.2.5.
- 1160 8. The `xrd:XRD/saml:Subject/saml:NameID` element MUST be present and equal to
1161 the `xrd:XRD/xrd:Query` element.
- 1162 9. The `NameQualifier` attribute of the
1163 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
1164 present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.
- 1165 10. There MUST be exactly one `saml:AttributeStatement` present in the
1166 `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute`
1167 element with a `Name` attribute of "`xri://$xrd*($v*2.0)`". This `saml:Attribute`
1168 element MUST contain exactly one `saml:AttributeValue` element whose text value
1169 is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that is the immediate
1170 parent of the `saml:Assertion` element.

Comment [DSR14]: TODO: confirm the Name attribute value (Peter)

1171 8.2.3 Recursing Authority Resolution

1172 If a resolver requests trusted resolution of multiple authority subsegments (see section 7.1.6), a
1173 recursing authority resolution service SHOULD attempt to perform trusted resolution on behalf of
1174 the resolver as described in this section. However if the resolution service is not able to obtain
1175 trusted XRDs for one or more additional recursing subsegments, it SHOULD return only the
1176 trusted XRDs it has obtained and allow the resolver to continue.

1177 8.2.4 Client Validation of XRDs

1178 For each XRD returned as part of a trusted resolution request, the resolver MUST validate the
1179 XRD according to the rules defined in this section.

- 1180 1. The `xrd:XRD/saml:Assertion` element MUST be present.
- 1181 2. This assertion MUST valid per the processing rules described by [SAML].
- 1182 3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by
1183 [XMLDSig] and constrained by Section 5.4 of [SAML].

- 1184 4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML
1185 assertion. Specifically, the signature MUST contain a single
1186 `ds:SignedInfo/ds:Reference` element, and the URI attribute of this reference
1187 MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent
1188 of the signed SAML assertion.
- 1189 5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo`
1190 element, the resolver MAY reject the signature if this key does not match the signer's
1191 expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor
1192 that was used to describe the current authority. See section 8.2.5.
- 1193 6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose
1194 resolution resulted in the current XRD.
- 1195 7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1196 `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability
1197 of trusted resolution service from this authority as required in section 8.2.5.
- 1198 8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1199 `NameQualifier` attribute of the
1200 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1201 9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the
1202 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1203 10. There MUST exist exactly one
1204 `xrd:XRD/saml:Assertion/saml:AttributeStatment` with exactly one
1205 `saml:Attribute` element that has a `Name` attribute of "`xri://$xrd*($v*2.0)`". This
1206 `saml:Attribute` element must have exactly one `saml:AttributeValue` element
1207 whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that
1208 is the immediate parent of the signed SAML assertion.

1209 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result
1210 MUST NOT be considered a valid trusted resolution response as defined by this specification.
1211 Note that this does not preclude a resolver from considering alternative resolution paths. For
1212 example, if an XRD advertising SAML trusted resolution service has two or more
1213 `xrd:XRD/xrd:Service/xrd:URI` elements and the response from one service endpoint fails
1214 to meet the requirements above, the client MAY repeat the validation process using the second
1215 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as
1216 defined by this document and SAML trusted resolution may continue.

1217 8.2.5 Correlation of ProviderID and KeyInfo Elements

1218 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at
1219 least one unique persistent service provider identifier expressed in the
1220 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority
1221 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI
1222 authority. A ProviderID may be any valid URI that meets these requirements of persistence and
1223 uniqueness. Examples of appropriate URIs include URNs as defined by [RFC2141] and fully
1224 persistent XRIs expressed in URI-normal form as defined by [XRISyntax].

1225 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in
1226 an XRD advertising trusted authority resolution service with the response received from a trusted
1227 resolution service endpoint. If the signed XRD response contains the same ProviderID as the
1228 XRD used to advertise a service, and the resolver has reason to trust the signature, the resolver
1229 can trust that the XRD response has not been maliciously replaced with another XRD.

1230 There is no defined discovery process for the ProviderID for a community root authority; it must
1231 be published in the community root XRDS document (or other equivalent description—see
1232 section 7.1.3) and verified independently. Once the community root XRDS document is known,

1233 the ProviderID for delegated XRI authorities within this community MAY be discovered using the
1234 `xrd:XRD/xrd:Service/xrd:ProviderID` element of authority resolution service endpoints.
1235 This trust mechanism may also be used for other services offered by an authority.

1236 In addition, the metadata necessary for SAML trusted authority resolution or other SAML [SAML]
1237 interactions MAY be discovered using the `ds:KeyInfo` element (section 4.2.) Again, if this
1238 element is present in an XRD advertising SAML authority resolution service (or any other
1239 service), and the client has reason to trust this XRD, the client MAY use the associated
1240 ProviderID to correlate the contents of this element with a signed response.

1241 To assist resolvers in using this key discovery mechanism, it is important that trusted authority
1242 resolution services be configured to sign responses in such a way that the signature can be
1243 verified using the correlated `ds:KeyInfo` element. For more information, see [SAML].

1244 8.3 HTTPS+SAML

1245 8.3.1 Service Type and Service Media Type

1246 The protocol defined in this section is identified by the values in Table 16.

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

1247 *Table 16: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.*

1248 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST
1249 use the Service Type identifier and Service Media Type identifier defined in Table 16 (including
1250 the `https=true` and `saml=true` parameters). In addition, the identifier authority MUST use an
1251 HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1252 8.3.2 Protocol

1253 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- 1254 1. All authority resolution service endpoints MUST be selected using the values defined in
1255 Table 16.
- 1256 2. All authority resolution requests and responses, including the starting request to a
1257 community root authority, MUST conform to both the requirements of the HTTPS trusted
1258 resolution protocol defined in section 8.1 and the SAML trusted resolution protocol
1259 defined in section 8.2.
- 1260 3. All authority resolution requests MUST contain an HTTPS Accept header with the media
1261 type identifier defined in Table 16 (including both the `https=true` and `saml=true`
1262 parameters). This MUST be interpreted as the value of the Resolution Output Format
1263 input parameter.
- 1264 4. If the resolver finds that an authority in the resolution chain does not support both HTTPS
1265 and SAML, the resolver MUST return a 23x error as defined in section 0.

1266

9 Proxy Resolution

1267

The preceding sections have defined XRI resolution as a set of logical functions that may implemented via a local resolver interface. This section defines a mapping of these functions to an HTTP(S) interface for remote invocation. This mapping is based on a standard syntax for expressing an XRI as an HTTP URI, called an *HXRI*, as defined in section 9.2. This includes a method of passing the other XRI resolution input parameters as query parameters in the HXRI.

1270

1271

Proxy resolution is useful for many reasons:

1272

- Offloading XRI resolution and service endpoint selection processing from a client to an HTTP(S) server.

1273

1274

- Optimizing XRD caching for a resolution community (a *caching proxy resolver*). Proxy resolvers SHOULD use caching to resolve the same QXRI or QXRI components for multiple clients as defined in section 15.4.

1275

1276

1277

- Returning HTTP(S) redirects to clients such as browsers that have no native understanding of XRIs but can process HXRIs. This provides backwards compatability with the large installed base of existing HTTP clients.

1278

1279

1280

9.1 Service Type and Media Types

1281

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

1282

Service Type	Service Media Types	Media Type Parameters
xri://\$res*proxy*(\$v*2.0)	application/xrds+xml application/xrd+xml text/uri-list	All parameters specified in Table 6

1283

Table 17: Service Type and Service Media Type values for proxy resolution.

1284

A proxy resolution service endpoint advertised in an XRDS document MUST use the Service Type identifier and Service Media Type identifiers defined in Table 17 (including the optional media type parameters). In addition, if the media type parameter `https=true` is included, the identifier authority MUST offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1285

1286

1287

1288

It may appear to be of limited value to advertise proxy resolution service in an XRDS document if a resolver must already know how to perform local XRI resolution in order to retrieve this document. However advertising a proxy resolution service in the XRDS document for a community root authority (sections 7.1.3 and 7.1.4) can be very useful for applications that need to consume XRI proxy resolution services or automatically generate HXRIs for resolution by non-XRI-aware clients in that community. Those applications may discover the current URI(s) and media type capabilities of a proxy resolver from this source.

1289

1290

1291

1292

1293

1294

1295

1296

9.2 HXRIs

1297

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

1298

1299

- It allows XRIs to be used anyplace an HTTP URI can appear, including in Web pages, electronic documents, email messages, instant messages, etc.

1300

1301

- It allows XRI-aware processors and search agents to recognize an HXRI and extract the embedded XRI for direct resolution, processing, and indexing.

1302

1303

Comment [DSR15]: In the future this section will move to XRI Syntax 3.0.

1304 To make this syntax as simple as possible for XRI-aware processors or search agents to
1305 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority segment that
1306 begins with the domain name segment "xri.". The QXRI is then appended as the entire local
1307 path (and query component, if present). The QXRI MUST NOT include the "xri://" prefix and
1308 MUST be in URI-normal form as defined in **[XRISyntax]**. (If a proxy resolver receives an HXRI
1309 containing a QXRI beginning with an "xri://" prefix, it SHOULD follow Postel's Law¹ by
1310 removing it before continuing.) In essence, the proxy resolver URI (including the forward slash
1311 after the domain name) serves as a machine-readable prefix for an absolute XRI in URI-normal
1312 form.

1313 The normative ABNF for an HXRI is defined below based on the *ireg-name*, *xri-hier-part*,
1314 and *iquery* productions defined in **[XRISyntax]**. Authors whose XRIs need to be understood by
1315 non-XRI-aware clients SHOULD publish them as HTTP URIs conforming to this HXRI production.

```
1316 HXRI           = proxy-resolver "/" QXRI
1317 proxy-resolver = ( "http://" / "https://" ) proxy-reg-name
1318 proxy-reg-name = "xri." ireg-name
1319 QXRI           = xri-hier-part [ "?" i-query ]
```

¹ http://en.wikipedia.org/wiki/Postel%27s_Law

1320 URI processors that recognize XRIs SHOULD interpret the local part of an HTTP or HTTPS URI
 1321 (the path segments and optional query segment) that conforms to this ABNF as an XRI provided
 1322 the first segment of the path conforms to the xri-authority or iauthority productions in
 1323 [XRISyntax].

1324 For references to communities that offer public XRI proxy resolution services, see the XRI
 1325 Technical Committee home page at <http://www.oasis-open.org/committees/xri>.

Comment [DSR16]: OPEN ISSUE: pending.

1326 9.3 HXRI Query Parameters

1327 There are a total of five logical input parameters to XRI authority resolution and service endpoint
 1328 selection as defined in section 6.1:

- 1329 1. QXRI – the entire original XRI for which resolution is requested.
- 1330 2. Path String – the path component of the QXRI.
- 1331 3. Resolution Output Format – the requested media type of the resolution response (see
 1332 section 6.1.2).
- 1333 4. Service Type – the type of service requested for service endpoint selection (see section
 1334 6.1.3).
- 1335 5. Service Media Type – the type of media requested for service endpoint selection (see
 1336 section 6.1.4).

1337 In proxy resolution, these parameters are bound to an HTTP(S) interface using the conventional
 1338 web model of encoding them in an HTTP(S) URI, which in this case is an HXRI. The binding of
 1339 the logical parameter names to HXRI component parts is defined in Table 18.

Logical Parameter Name	HXRI Component	HXRI Query Parameter Name
QXRI	Entire path and query string of HXRI	N/A
Path String	All segments of the HXRI path except the first segment	N/A
Resolution Output Format	HXRI query parameter	_xrd_r
Service Type	HXRI query parameter	_xrd_t
Service Media Type	HXRI query parameter	_xrd_m

1340 *Table 18: Binding of logical XRI resolution parameters to QXRI query parameters.*

1341 Following are the rules for the use of the parameters specified in Table 18.

- 1342 1. The QXRI MUST be normalized as specified in section 9.2.
- 1343 2. If the original QXRI has an existing query component, the HXRI query parameters MUST
 1344 be appended to that query component. (Note that the query parameter names in Table
 1345 18 were chosen to minimize the probability of collision with any other existing query
 1346 parameter names.) After proxy resolution, the HXRI query parameters MUST
 1347 subsequently be removed from the QXRI query component. The existing QXRI query
 1348 component MUST NOT be altered in any other way, i.e., it must be passed through with
 1349 no changes in parameter order, escape encoding, etc.
- 1350 3. If the original QXRI does not have a query component, one MUST be added to pass any
 1351 HXRI query parameters. After proxy resolution, this query component MUST be entirely
 1352 removed.

- 1353 4. If the original QXRI had a null query component (only a leading question mark), or a
1354 query component consisting of only question marks, *one additional leading question mark*
1355 **MUST** be added before adding any HXRI query parameters. After proxy resolution, any
1356 HXRI query parameters and exactly one leading question mark **MUST** be removed. See
1357 the URI construction step defined in section 10.6.
- 1358 5. Each HXRI query parameter **MUST** be delimited from other parameters by an ampersand
1359 (“&”). Any occurrences of the character “&” within an input parameter (specifically the
1360 Service Type value) **MUST** be percent encoded prior to input and decoded upon output.
- 1361 6. Each HXRI query parameter **MUST** be delimited from its value by an equals sign (“=”).
- 1362 7. In an HXRI query parameter, the character + and the percent-encoded sequence %2B
1363 **MUST** be interpreted as the same character, therefore spaces in an HXRI query
1364 parameter **MUST NOT** be encoded as + signs.
- 1365 8. Any XRIs or IRIs used as values of HXRI query parameters **MUST** be in URI-normal form
1366 as defined in **[XRISyntax]**.
- 1367 9. If any HXRI query parameter name is included but its value is empty, the value of the
1368 parameter **MUST** be considered null.
- 1369 10. For proxy resolution, any input parameter supplied explicitly via an HXRI query parameter
1370 **MUST** take precedence over the same parameter provided via an HTTP(S) header, even
1371 if the value of the HXRI query parameter is explicitly null. See the following section.

1372 9.4 HTTP(S) Accept Headers

1373 In proxy resolution, one XRI resolution input parameter, the Service Media Type (section 6.1.4)
1374 **MAY** be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. The
1375 following rules apply to this input:

- 1376 1. As described in section 14.1 of **[RFC2616]**, the Accept header content type **MAY** consist
1377 of multiple media type identifiers. If so, the proxy resolver **MUST** choose only one to
1378 accept. A proxy resolver client **SHOULD** order media type identifiers according to the
1379 client’s preference and a proxy resolver server **SHOULD** choose the client’s highest
1380 preference.
- 1381 2. If the value of the Accept header content type is null, this **MUST** be interpreted as the
1382 value of the Service Media Type parameter.
- 1383 3. If the value of the Service Media Type parameter is explicitly set via the `_xrd_m` query
1384 parameter in the HXRI (including to a null value), this **MUST** take precedence over any
1385 value set via an HTTP(S) Accept header.

1386 9.5 Null Resolution Output Format

1387 Unlike authority resolution as defined in the preceding sections, a proxy resolver **MAY** receive a
1388 resolution request where the Resolution Output Format input parameter value is null—either
1389 because this parameter is absent or because it was explicitly set to null using the `_xrd_r` query
1390 parameter.

1391 If the value of the Resolution Output Format value is null, a resolver **MUST** act as if the following
1392 media type parameters had the following values: `https=false`, `saml=false`, `refs=true`,
1393 `sep=true`, and `nodefault=false`. In addition, the output **MUST** be an HTTP(S) redirect as
1394 defined in the following section.

1395 9.6 Outputs and HTTP(S) Redirects

1396 For all values of the Resolution Output Format parameter except null, a proxy resolver **MUST**
1397 follow the output rules defined in section 6.2.

Comment [DSR17]: This section was completely rewritten in WD11 ED01 (see issue #41).

1398 If the value of the Resolution Output Format is null, and the output is not an error, a proxy
1399 resolver MUST follow the rules for output of a URI List as defined in section 6.2.3. However
1400 instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as
1401 an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service
1402 Media Type parameter.

1403 If the output is an error, a proxy resolver SHOULD return a human-readable error message with a
1404 media type of either `text/plain` or `text/html`.

1405 This rule enables XRI proxy resolvers to serve clients that do not understand XRI syntax or
1406 resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the
1407 service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept
1408 header (if any) as described in section 9.4.

1409 **9.7 Differences Between Proxy Resolution Servers**

1410 An XRI proxy resolution request MAY be sent to any proxy resolver that will accept it. All XRI
1411 proxy resolvers SHOULD deliver uniform responses given the same QXRI and other input
1412 parameters. However because proxy resolvers may potentially need to make decisions about
1413 network errors, reference processing, and trust policies on behalf of the client they are proxying,
1414 and these decisions may be based on local policy, in some cases different proxy resolvers may
1415 return different results.

1416 **9.8 Combining Authority and Proxy Resolution Servers**

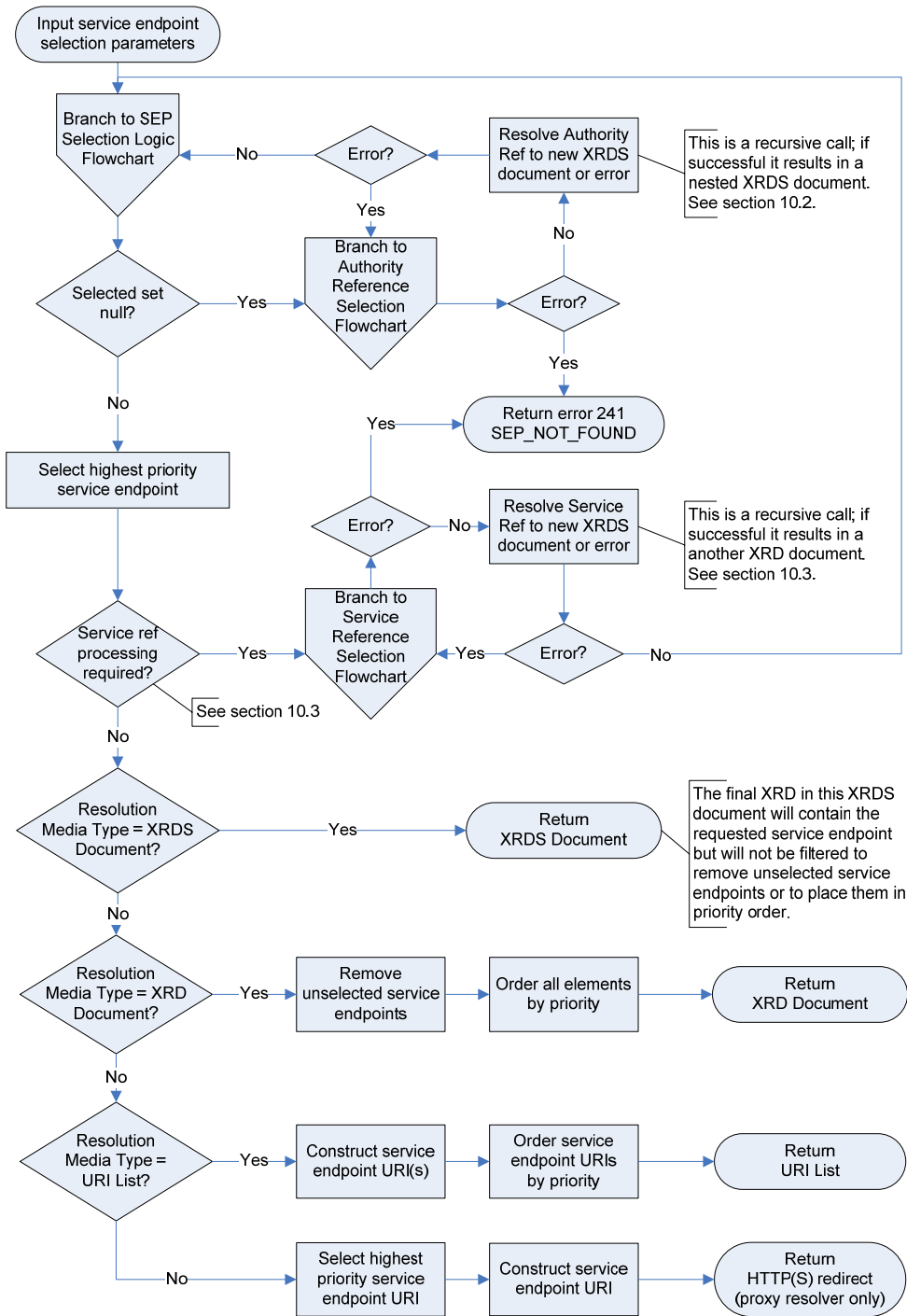
1417 The majority of DNS nameservers are recursing nameservers that answer both queries for which
1418 they are authoritative and queries which they must forward to other nameservers. The same
1419 applies to XRI architecture: in many cases the optimum configuration will be to combine an
1420 authority resolution service and a proxy resolution service in the same server. This server can
1421 publish a self-describing XRDS document (section 7.1.4) that advertises both its authority
1422 resolution and proxy resolution service endpoints. It can also optimize caching of XRDS for clients
1423 in its resolution community (see section 15.4).

1424 **10 Service Endpoint Selection**

1425 If the authority resolution phase is successful, the output is an XRDS document containing a final
1426 XRD describing the target authority. If requested, a resolver may perform an optional second
1427 phase of XRI resolution by processing this XRDS document to select a requested service
1428 endpoint. This section specifies the rules for this process.

1429 **10.1 Processing Rules**

1430 Figure 3 (non-normative) shows the overall logical flow of the service endpoint selection process.



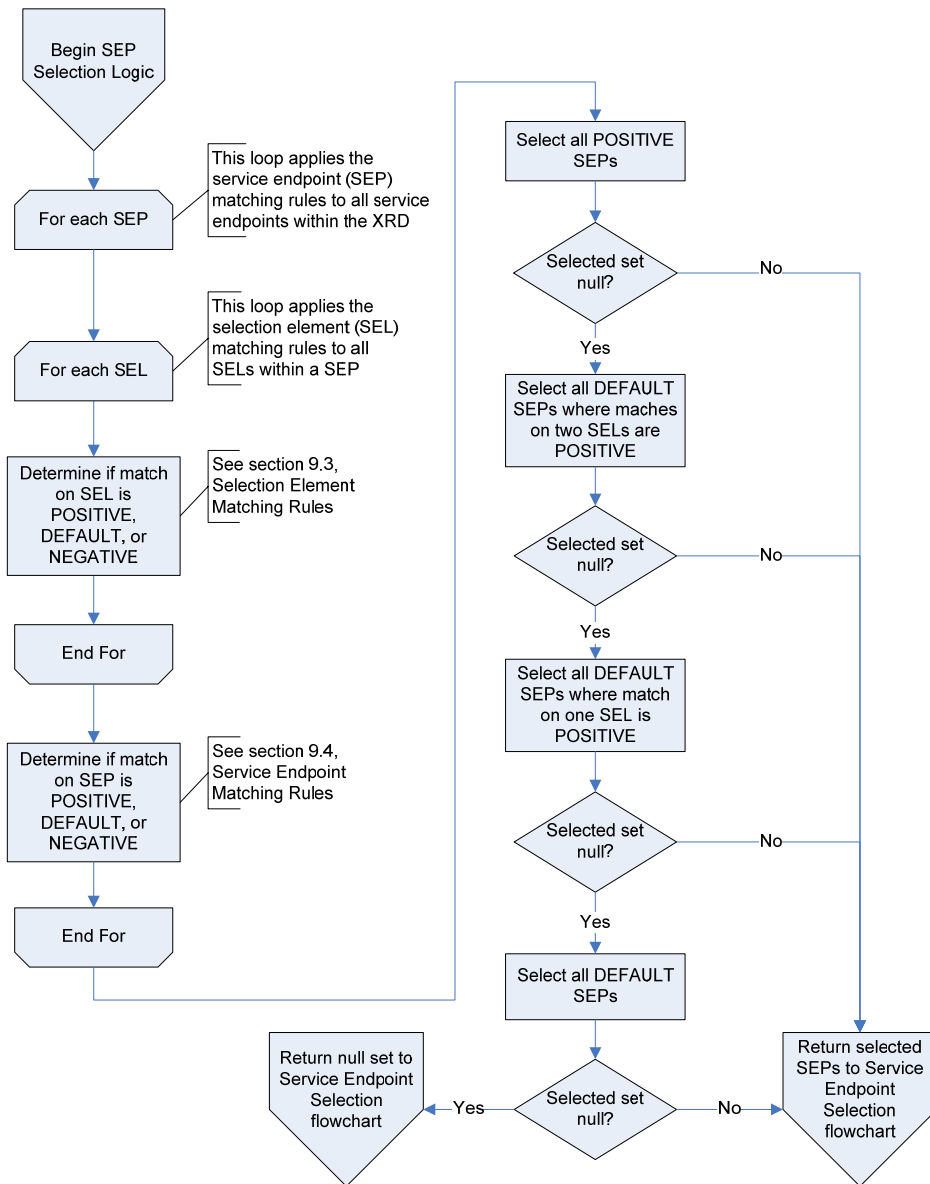
1431
1432

Figure 3: Service endpoint selection flowchart.

- 1433 Following are the normative rules for the overall service endpoint selection process:
- 1434 1. The inputs for service endpoint selection are defined in Table 7.
- 1435 2. For each `xrd:Service` element in the current XRD, selection MUST follow the service
1436 endpoint selection logic defined in section 10.2. The outcome of this process will be a
1437 selected set of zero or more service endpoints.
- 1438 3. If after applying the service endpoint selection rules no service endpoint is selected, a
1439 resolver MUST perform authority reference processing as defined in section 13.1 if the
1440 `refs` media type parameter is set to `true` or null, or if the parameter is absent. If the
1441 `refs` media type parameter is set to `false` and the XRD contains at least one
1442 `xrd:XRD/xrd:Ref` element that could be followed, the resolver MUST return a
1443 successful response with a status code of 101 `REF_NOT_FOLLOWED`.
- 1444 4. If an XRD does not include the requested service endpoint and does not include any
1445 `xrd:XRD/xrd:Ref` elements, the resolver MUST return an error with a status code of
1446 241 `SEP_NOT_FOUND` regardless of the value of the `refs` media type parameter.
- 1447 5. If after applying the service endpoint selection rules at least one service endpoint is
1448 selected, but it does NOT contain an `xrd:XRD/xrd:Service/xrd:URI` element and
1449 DOES contain an `xrd:XRD/xrd:Service/xrd:Ref` element, a resolver MUST
1450 perform service reference processing as defined in section 13.2.
- 1451 6. The output of service endpoint selection MUST be a valid Resolution Output Format as
1452 defined in Table 9. If this output is an XRDS Document, an XRD Document, or a URI List,
1453 it MUST conform to the output requirements defined in section 6.2. If the Resolution
1454 Output Format value is null and the output is an HTTP(S) redirect, the resolver MUST
1455 proceed as defined in section 9.6.

1456 10.2 Service Endpoint Selection Logic

1457 Selection of service endpoints (SEPs) within an XRD is managed using service endpoint
1458 selection elements (SEs). As defined in section 4.2.7, `xrd:XRD/xrd:Service` elements may
1459 contain three categories of selection elements: `xrd:Type`, `xrd:Path`, and `xrd:MediaType`.
1460 Figure 3 (non-normative) shows the overall flow of the service endpoint selection logic.



1461

1462 *Figure 4: Service endpoint (SEP) selection logic flowchart.*

1463 The following sections provide the normative rules for each section of this flowchart.

1464 **10.3 Selection Element Matching Rules**

1465 The first set of rules govern the matching of selection elements.

1466 **10.3.1 Selection Element Match Options**

1467 Within each service endpoint, there is a match option for each of the three categories of selection
1468 elements. The three match options are defined in Table 19:

Match Option	Match Condition	Precedence
POSITIVE	A successful match based on the value of the <code>xrd:match</code> attribute as defined in 10.3.2 OR a successful match based the contents of the selection element as defined in sections 10.3.6 - 10.3.8.	1
DEFAULT	The value of the <code>xrd:match</code> attribute is <code>default</code> OR there is no instance of this type of selection element contained in the service endpoint as defined in section 10.3.3.	0
NEGATIVE	The selection element does not satisfy either condition above.	-1

1469 *Table 19: Match options for selection elements.*

1470 **10.3.2 The Match Attribute**

1471 All three service endpoint selection elements accept the optional `xrd:match` attribute. This
1472 attribute gives XRDS authors precise control over selection of service endpoints based on the
1473 QXRI and other resolution input parameters. An enumerated list of the values for this attribute is
1474 defined in Table 20. If this attribute is present with one of these values, the contents of the
1475 selection element MUST be ignored, and the corresponding matching rule MUST be applied.

Value	Matching Rule Applied to Corresponding Input Parameter
any	Automatically a POSITIVE match (i.e., input parameter is ignored).
default	Automatically a DEFAULT match (i.e., input parameter is ignored) UNLESS the value of the Resolution Output Format <code>nodefault</code> parameter is set to <code>true</code> for this category of selection element, in which case it is a NEGATIVE match.
non-null	Any input value except null is a POSITIVE match. An input value of null is a NEGATIVE match.
null	An input value of null is a POSITIVE match. Any other input value is a NEGATIVE match.

1476 *Table 20: Enumerated values of the global match attribute and corresponding matching rules.*

1477 Backwards Compatability Note: earlier working drafts of this specification included the values
1478 `match="none"` and `match="contents"`. Both are deprecated. The former is no longer
1479 supported and the latter is now the default behaviour of any selection element that does not
1480 include the match attribute or the value of this attribute is empty.

1481 **10.3.3 Absent Selection Element Matching Rule**

1482 If a service endpoint does not contain at least one instance of a particular category of selection
1483 element, it MUST be considered equivalent to the service endpoint having a DEFAULT match on
1484 that category of selection element.

1485 10.3.4 Empty Selection Element Matching Rule

1486 If a selection element is present in a service endpoint but the element is empty, and if the element
1487 does not contain an `xrd:match` attribute or the value of this attribute is empty, it MUST be
1488 considered equivalent to having a DEFAULT match on this selection element.

1489 10.3.5 Multiple Selection Element Matching Rule

1490 Each service endpoint has only one match option for each category of selection element.
1491 Therefore if a service endpoint contains more than one instance of the same category of selection
1492 element (i.e., more than one `xrd:Type`, `xrd:Path`, or `xrd:MediaType` element), matching
1493 MUST be based on the selection element with the highest precedence match option as defined in
1494 Table 19.

1495 10.3.6 Type Element Matching Rules

1496 The following rules apply to matching the value of the input Service Type parameter with the
1497 contents of a non-empty `xrd:XRD/xrd:Service/xrd:Type` element when its `xrd:match`
1498 attribute is absent or empty.

- 1499 1. The values of the Service Type parameter and the `xrd:XRD/xrd:Service/xrd:Type`
1500 element SHOULD be normalized according to the requirements of their identifier scheme
1501 prior to input. In particular, if an XRI, IRI, or URI does not include a local part (a path
1502 and/or query component), a trailing forward slash MUST be assumed after the authority
1503 component. In all other cases, a trailing forward slash MUST NOT be assumed, and
1504 therefore is significant in comparisons. In addition, if the value is an XRI or an IRI it
1505 MUST be in URI-normal form as defined in section 4.4. XRI resolvers MAY perform
1506 normalization of these values but MUST NOT be required to do so. As a best practice,
1507 service designers SHOULD assign identifiers for service types that are easy to match
1508 and do not require normalization.
- 1509 2. To result in a POSITIVE match, the values MUST be equivalent according to the
1510 equivalence rules of the applicable identifier scheme. Any other result is a NEGATIVE
1511 match.

1512 10.3.7 Path Element Matching Rules

1513 The following rules apply to matching the value of the input Path String with the contents of a
1514 non-empty `xrd:XRD/xrd:Service/xrd:Path` element when its `xrd:match` attribute is
1515 absent or empty.

- 1516 1. If the value is a relative XRI or an IRI it MUST be in URI-normal form as defined in
1517 section 4.4.
- 1518 2. Trailing forward slashes MUST be significant in comparisons.
- 1519 3. [TODO: OPEN ISSUE] Equivalence comparison MUST be performed using Caseless
1520 Matching as defined in section 3.13 of [Unicode], with the exception that it is not
1521 necessary to perform normalization after case folding.
- 1522 4. If the value of the Path String is a *subsegment stem match* with the contents of the Path
1523 element, this MUST be a POSITIVE match. Any other result MUST be a NEGATIVE
1524 match. A subsegment stem match is defined as the entire Path String being character-
1525 for-character equivalent with any continuous sequence of subsegments or segments
1526 (including empty subsegments and empty segments) in the contents of the Path element
1527 beginning from the most significant (leftmost) subsegment.

1528 Examples of the this rule are shown in table [TODO] below.

Comment [DSR18]: TODO - OPEN ISSUE

1529 10.3.8 MediaType Element Matching Rules

1530 The following rules apply to matching the value of the input Service Media Type parameter with
1531 the contents of a non-empty `xrd:XRD/xrd:Service/xrd:MediaType` element when its
1532 `xrd:match` attribute is absent or empty.

- 1533 1. The values of the Service Media Type parameter and the `xrd:MediaType` element
1534 SHOULD be normalized according to the rules for media types in section 3.7 of
1535 [RFC2616] prior to input. (The rules are that type and subtype names are case-
1536 insensitive, but parameter values may or may not be case-sensitive depending on the
1537 semantics of the parameter name. XRI Resolution Output Format parameters are case-
1538 insensitive.) XRI resolvers MAY perform normalization of these values but MUST NOT be
1539 required to do so.
- 1540 2. To be a POSITIVE match, the values MUST be character-for-character equivalent. Any
1541 other result is a NEGATIVE match.

1542 10.4 Service Endpoint Matching Rules

1543 The next set of matching rules govern the matching of service endpoints based on the match
1544 types of the selection elements they contain.

1545 10.4.1 Service Endpoint Match Options

1546 For each service endpoint in an XRD, there are three match options as defined in Table 21:

Match Option	Condition
POSITIVE	Meets the Select Attribute Match Rule (section 10.4.2) or the All Positive Match Rule (section 10.4.3).
DEFAULT	Meets the Default Match Rule (section 10.4.4).
NEGATIVE	The service endpoint does not satisfy either condition above.

1547 *Table 21: Match options for service endpoints.*

1548 10.4.2 Select Attribute Match Rule

1549 All three service endpoint selection elements accept the optional `xrd:select` attribute. This
1550 attribute is a Boolean value used to govern selection of the containing service endpoint according
1551 to the following rule. If service endpoint contains a selection element with a POSITIVE match as
1552 defined in section 10.3, and the value of this selection element's `xrd:select` attribute is `true`,
1553 the service endpoint automatically MUST be a POSITIVE match, i.e., all other selection elements
1554 for this service endpoint MUST be ignored.

1555 10.4.3 All Positive Match Rule

1556 If a service endpoint has a POSITIVE match on all three categories of selection elements
1557 (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) as defined in section 10.3, the service endpoint
1558 MUST be a POSITIVE match. If even one of the three selection element match types is not
1559 POSITIVE, this rule fails.

1560 10.4.4 Default Match Rule

1561 If a service endpoint fails the All Positive Match Rule, but none of the three categories of selection
1562 elements has a NEGATIVE match as defined in section 10.3, the service endpoint MUST be a
1563 DEFAULT match.

1564 10.5 Service Endpoint Selection Rules

1565 The final set of rules governs the selection of service endpoints based on their match types.

1566 10.5.1 Positive Match Rule

1567 After applying the matching rules to service endpoints in section 10.4, all service endpoints that
1568 have a POSITIVE match MUST be selected. Only if there are no service endpoints with a
1569 POSITIVE match is the Default Match Rule invoked.

1570 10.5.2 Default Match Rule

1571 If there are no service endpoints with a POSITIVE match, then the service endpoints with a
1572 DEFAULT match that have the highest number of POSITIVE matches on each category of
1573 selection element MUST be selected. This means:

- 1574 1. The service endpoints in the DEFAULT set that have two POSITIVE selection element
1575 matches MUST be selected.
- 1576 2. If the previous set is empty, the service endpoints in the DEFAULT set that have one
1577 POSITIVE selection element match MUST be selected.
- 1578 3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.
- 1579 4. If the previous set is empty, no service endpoint is selected and the return set is null.

1580 10.6 Pseudocode

1581 The following pseudocode provides a precise description of the service endpoint selection logic.
1582 The pseudocode is normative, however if there is a conflict between it and the rules stated in the
1583 preceding sections, the preceding sections shall prevail.

1584 The pseudocode uses nine Boolean flags to record the match state for each category of selection
1585 element (SEL) in a service endpoint (SEP):

- 1586 • Positive.Type
- 1587 • Positive.Path
- 1588 • Positive.Mediatype
- 1589 • Default.Type
- 1590 • Default.Path
- 1591 • Default.Mediatype
- 1592 • Matched.Type
- 1593 • Matched.Path
- 1594 • Matched.Mediatype

1595

```
1596 FOR EACH SEP
1597   SET all flags to FALSE
1598   FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
1599     SET Matched.x=TRUE
1600     IF match is POSITIVE
1601       IF select="true"           ;see section 9.4.2
1602         ADD SEP TO SELECTED SET
1603       NEXT SEP
1604     ELSE
1605       SET Positive.x=TRUE
1606     NEXT SEL
1607   ENDIF
```

```

1608     ELSEIF match is DEFAULT
1609         IF Positive.x=TRUE
1610             NEXT SEL
1611         ELSEIF ndefault="x"           ;see section 9.3.2
1612             NEXT SEL
1613         ELSE
1614             SET Default.x=TRUE
1615             NEXT SEL
1616         ENDIF
1617     ELSEIF match is NEGATIVE
1618         NEXT SEL
1619     ENDIF
1620 ENDFOR
1621 IF Matched.x=FALSE
1622     SET Default.x=TRUE
1623 ENDIF
1624 IF Positive.Type=TRUE AND
1625     Positive.Path=TRUE AND
1626     Positive.MediaType=TRUE
1627     ADD SEP TO SELECTED SET
1628     NEXT SEP
1629 ELSEIF SELECTED SET != EMPTY           ;test if defaults are needed
1630     NEXT SEP
1631 ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
1632     (Positive.Path=TRUE OR Default.Path=TRUE) AND
1633     (Positive.MediaType=TRUE OR Default.MediaType=TRUE)
1634     ADD SEP TO DEFAULT SET
1635 ENDIF
1636 ENDFOR
1637 IF SELECTED SET != EMPTY
1638     RETURN SELECTED SET
1639 ELSEIF DEFAULT SET = EMPTY
1640     RETURN DEFAULT SET
1641 ENDIF
1642 FOR EACH SEP IN DEFAULT SET
1643     IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
1644         (Positive.Type=TRUE AND Positive.MediaType=TRUE) OR
1645         (Positive.Path=TRUE AND Positive.MediaType=TRUE)
1646         ADD SEP TO SELECTED SET
1647     ENDIF
1648 ENDFOR
1649 IF SELECTED SET != EMPTY
1650     RETURN SELECTED SET
1651 ENDIF
1652 FOR EACH SEP IN DEFAULT SET
1653     IF Positive.Type=TRUE OR
1654         Positive.Path=TRUE OR
1655         Positive.MediaType=TRUE
1656         ADD SEP TO SELECTED SET
1657     ENDIF
1658 ENDFOR
1659 IF SELECTED SET != EMPTY
1660     RETURN SELECTED SET
1661 ELSE
1662     RETURN DEFAULT SET
1663 ENDIF

```

1664 10.7 Construction of Service Endpoint URIs

1665 If the output is a URI List, the final step in the service endpoint selection process is construction
1666 of the service endpoint URI(s). This is governed by the `xrd:append` attribute of each `xrd:URI`
1667 element. The values of this attribute are shown in Table 22.

Value	Component of QXRI to Append
none	None. This is the default if the <code>xrd:append</code> attribute is absent or its value is empty.
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 “/”).
authority	Authority string only (including the community root subsegment).
path	Path string only (including the leading “/”).
query	Query string only (including the leading “?”).
qxri	Entire QXRI.

1668 Table 22: Values of the `append` attribute and the corresponding QXRI component to append.

1669 If the `xrd:append` attribute is absent or its value is empty, the default value is `none`. Following
1670 are the rules for construction of the final service endpoint URI based on the value of the
1671 `xrd:append` attribute. Note that these rules must be followed closely in order to give XRD
1672 authors precise control over construction of service endpoint URIs.

- 1673 1. If the value is `none`, the exact contents of the `xrd:URI` element MUST be returned
1674 directly without any further processing.
- 1675 2. For any other value, the exact value of the QXRI component specified in Table 22,
1676 including any leading delimiter(s), with no additional escaping or percent encoding MUST
1677 be appended directly to the exact contents of the `xrd:URI` element including any trailing
1678 delimiter(s). If the value of the QXRI component specified in Table 22 consists of only a
1679 leading delimiter, then this value MUST be appended according to these rules. If the
1680 value of the QXRI component specified in Table 22 is null, then the contents of the
1681 `xrd:URI` element MUST be returned directly exactly as if the value of the `xrd:append`
1682 attribute was `none`.
- 1683 3. If any query parameters for XRI proxy resolution were added to an existing QXRI query
1684 component as defined in section 9.3, these query parameters MUST be removed prior to
1685 performing the append operation as also defined in section 9.3. In particular, if after
1686 removal of these query parameters the QXRI query component consists of only a string
1687 of one or more question marks (the delimiting question mark plus zero or more additional
1688 question marks) then exactly one question mark MUST also be removed. This preserves
1689 the query component of the original QXRI if it was null or contained only question marks.

1690 **IMPORTANT:** Construction of HTTP(S) URIs for XRI authority resolution service endpoints is
1691 defined in section 7.1.8. Note that this involves an additional step taken after all URI construction
1692 processing in this section is complete. In other words, if the URI element of an XRI authority
1693 resolution service endpoint includes an `append` attribute, the Next Authority Service URI should
1694 be fully constructed according to the algorithm in this section before appending the Next Authority
1695 String as defined in section 7.1.8.

1696

11 Synonyms

1697
1698
1699

XRI resolution includes support for *synonyms*—XRIs, IRIs, or URIs that are not character-for-character equivalent, but which identify the same target resource (in the same context, or across different contexts). Table 23 describes the five types of synonyms supported in XRDs.

Synonym Element	Cardinality	Synonym Type	Synonym Scope	Resolves to different XRD?
LocalID	Zero-or-more	Hierarchical	Relative	No
GlobalID	Zero-or-more	Hierarchical	Absolute	No
Ref	Zero-or-more	Polyarchical	Absolute	Yes
Backref	Zero-or-more	Polyarchical	Absolute	No
CanonicalID	Zero-or-one	Hierarchical or polyarchical	Absolute	No if hierarchical, Yes if polyarchical

1700

Table 23: The five XRD synonym elements.

1701
1702

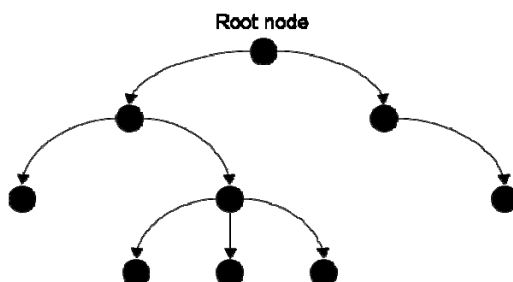
This section explains the differences between hierarchical and polyarchical synonyms and specifies the semantics and recommended usage of each type of synonym.

1703

11.1 Hierarchical and Polyarchical Synonyms

1704
1705

A hierarchical identifier system is a graph of parent-child relationships in which every child node is identified by exactly one arc from exactly one parent node.



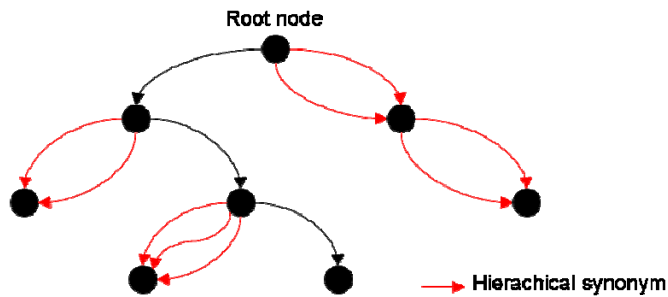
1706

1707

Figure 5: Hierarchical identifier graph.

1708
1709
1710
1711

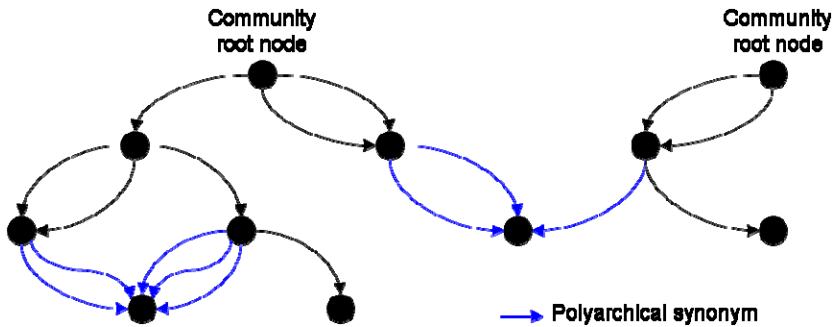
A hierarchical identifier system can support synonyms by changing the rule to allow a child node to have one-or-more arcs from exactly one parent node as shown in Figure 6. Each arc from the same parent to the same child is a *hierarchical synonym* of the other arcs between that parent and child.



1712

1713 *Figure 6: Hierarchical identifier graph with synonyms.*

1714 A polyarchical identifier graph takes the next step: a child node may have *one-or-more arcs* from
 1715 *one-or-more* parent nodes. This means a child node may be a member of more than one
 1716 hierarchy. This reflects identification in the real world, where many resources have more than one
 1717 identifier in more than one context.

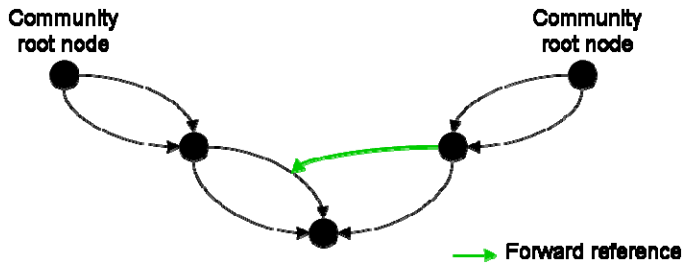


1718

1719 *Figure 7: Polyarchical identifier graph*

1720 In a polyarchical graph model, from the perspective of any particular parent node, the set of arcs
 1721 from it to the same child node are all hierarchical synonyms for that child, and the parent is auth-
 1722 oritative for all such arcs. From the perspective of the same parent node, however, the set of arcs
 1723 from *other* parent nodes to that child node are all *polyarchical synonyms* for that child—identifiers
 1724 for the same target resource, but for which this particular parent node is *not* authoritative.

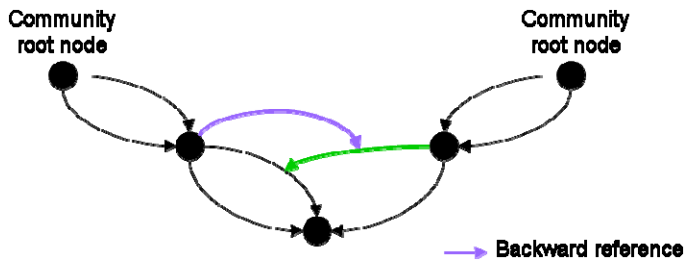
1725 Polyarchical synonyms are expressed as *arcs to arcs*, i.e., an arc that points to another arc
 1726 (called a child arc) instead of a child node. When two parent nodes share a child node, and one
 1727 has an arc pointing to a child arc from the other, this is called a *forward reference*.



1728

1729 *Figure 8: Polyarchical synonym expressing a forward reference.*

1730 If a parent needs to identify that another parent is using one of its arcs as a forward reference, it
 1731 requires an *arc to an arc to an arc* as shown in Figure 9. This form of polyarchical synonym is
 1732 called a *backward reference*.

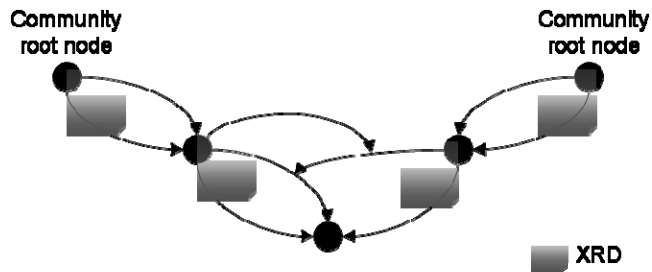


1733

1734 *Figure 9: Polyarchical synonym expressing a backward reference.*

1735 Forwards and backwards references are particularly useful in establishing and verifying trust
 1736 relationships spanning multiple hierarchies or trust domains. These are covered in more detail in
 1737 section 12, *Synonym Verification*.

1738 Since XRI architecture is based on a polyarchical graph model, every XRD represents the
 1739 perspective of a *particular parent node* about a *particular child node*. Each synonym element in
 1740 the XRD represents an arc from the parent node to the child node—either directly (a hierachical
 1741 synonym), or via another child arc (a polyarchical synonym).



1742

1743 *Figure 10: Each XRD expresses the set of arcs from a parent to a child.*

1744 Note that whether a given synonym is hierachical or polyarchical depends on the XRD in which it
 1745 appears. A synonym that is hierachical in an XRD from one parent is polyarchical in an XRD
 1746 from a different parent and vice versa.

1747 In the following sections, a parent node in the XRI polyarchical graph model is called a *parent*
 1748 *authority* and a child node is called a *child authority*. Note that in an XRD, the parent authority is
 1749 itself identified by the `xrd:XRD/xrd:ProviderID` element (if present). See section 4.2.

1750 11.2 LocalID

1751 A LocalID is a hierachical synonym expressed using the `xrd:LocalID` element. LocalIDs are
 1752 used at both the authority level of an XRD (as a child of the root `xrd:XRD` element) and the
 1753 service level (as a child of the root `xrd:XRD/xrd:Service` element).

1754 At the authority level, the value of the `xrd:XRD/xrd:LocalID` element is a relative identifier
 1755 that is interchangeable with the contents of the `xrd:Query` element. In other words, it **MUST** be
 1756 assigned by the same authority producing the current XRD and **MUST** identify the same child
 1757 authority as the the `xrd:Query` element. Resolution of a LocalID **MUST** return the same XRD as
 1758 the XRD in which it appears (with the exception of the values of the `xrd:Query`, `xrd:Expires`,
 1759 and `xrd:XRD/xrd:LocalID` elements).

1760 If an parent authority has assigned a child authority a persistent identifier along with one or more
 1761 reassignable identifiers, the authority **SHOULD** return the persistent identifier as a
 1762 `xrd:XRD/xrd:LocalID` value in any XRD returned for a reassignable identifier. The reverse

1763 MAY also be true, however authorities MAY adopt privacy or other policies that restrict the
1764 reassignable synonyms returned for any particular resolution request.

1765 A parent authority SHOULD NOT permit a child authority to edit a LocalID value in an XRD
1766 without authenticating the child authority and verifying that the child authority is authorized to use
1767 this LocalID value.

1768 At the service level, the `xrd:XRD/xrd:Service/xrd:LocalID` element MAY be used to
1769 express either a relative or absolute identifier for the child authority in the context of the specific
1770 service being described. If present, consuming applications SHOULD use the value of this
1771 element to identify the child authority in the context of this service endpoint. If not present,
1772 consuming applications SHOULD use the value of the `xrd:XRD/xrd:CanonicalID` element,
1773 or if not present, the value of the `xrd:XRD/xrd:GlobalID` element, to identify the child
1774 authority in the context of this service endpoint.

1775 **11.3 GlobalID**

1776 A GlobalID is a hierarchical synonym expressed using the `xrd:GlobalID` element. Like a
1777 LocalID, it MUST be assigned by the parent authority producing the XRD in which it appears.
1778 Unlike a LocalID, a GlobalID MUST be an absolute identifier for the child authority.

1779 Also like a LocalID, resolution of a GlobalID MUST return the same XRD as the XRD in which it
1780 appears (with the exception of the values of the `xrd:Query`, `xrd:Expires`, and
1781 `xrd:XRD/xrd:LocalID` elements).

1782 A GlobalID is the only hierarchical synonym element that uniquely identifies a child authority in a
1783 global scope (as described below, Refs and Backrefs must be polyarchical, and a CanonicalID
1784 may be either hierarchical or polyarchical). For this reason GlobalIDs play a special role in
1785 synonym verification, and a GlobalID SHOULD NOT be trusted unless it is verified as defined in
1786 section 12. A parent authority SHOULD NOT permit a child authority to edit a GlobalID value in
1787 an XRD without authenticating the child authority and verifying that the child authority is
1788 authorized to use this GlobalID value.

1789 Identifier authorities SHOULD publish a GlobalID for any child authority that may need a
1790 polyarchical CanonicalID at some point in its lifetime.

1791 **11.4 Ref (Forward Reference)**

1792 A Ref is a polyarchical synonym expressed using the `xrd:Ref` element. Like a LocalID, a Ref
1793 can be used at both the authority level of an XRD (as a child of the root `xrd:XRD` element) and
1794 the service level (as a child of the root `xrd:XRD/xrd:Service` element).

1795 At the authority level, a Ref asserts a forward reference as defined in section 11.1. This is an
1796 assertion by the parent authority assigning the Ref that another parent authority is authorized to
1797 describe their mutual child, and that a resolver can consider the XRDs from both authorities as a
1798 "logical union". A Ref MUST NOT be used to assert any other relationship between the
1799 authorities.

1800 An `xrd:XRD/xrd:Ref` element MUST be an absolute identifier and MUST be assigned by a
1801 different parent authority that the parent authority producing the XRD in which the Ref appears. If
1802 resolved, it MUST produce a different XRD than the XRD in which the Ref appears. See section
1803 13.1 for authority reference processing rules.

1804 At the service level, a service Ref MUST be an absolute HTTP(S) URI that directly references
1805 another XRDS document describing the target authority in the context of a specific service. See
1806 section 13.2 for service reference processing rules.

1807 **11.5 Backref (Backward Reference)**

1808 A Backref is a polyarchical synonym expressed using the `xrd:BackRef` element. A Backref
1809 asserts a backwards reference as defined in section 11.1. A backwards reference is the inverse
1810 of a forward reference, i.e., it represents a backpointer to an XRD that contains a Ref or a
1811 CanonicalID that resolves the current XRD.

1812 While Backrefs may be used for polyarchical synonym discovery, they are not intended for
1813 reference processing as defined in section 13. Their primary purpose is synonym authorization. A
1814 parent authority **MUST** use a Backref to authorize a Ref or a polyarchical CanonicalID synonym
1815 asserted by a different parent authority as defined in section 12. A parent authority **SHOULD NOT**
1816 permit a child authority to edit a Backref value in an XRD without authenticating the child authority
1817 and verifying that the child authority is authorized to use this Backref value.

1818 **11.6 CanonicalID**

1819 A CanonicalID is either a hierarchical or polyarchical synonym expressed using the
1820 `xrd:CanonicalID` element. It **MUST** be an absolute identifier. From the perspective of the
1821 parent authority providing the XRD, it represents the recommended synonym among all
1822 synonyms for the child authority. Parent authorities **SHOULD** publish a CanonicalID to make it
1823 easy for applications to discover the canonical identifier they should use to reference a target
1824 resource.

1825 For durability, a CanonicalID **SHOULD** be a persistent identifier such as a URN [**RFC2141**] or a
1826 persistent XRI [**XRISyntax**]. If a persistent CanonicalID value is present, it is **RECOMMENDED**
1827 that applications use it as a persistent identifier for the target resource and treat all other
1828 reassignable identifiers as temporary synonyms.

1829 Resolution of a hierarchical CanonicalID **MUST**, like a GlobalID, return the same XRD as the XRD
1830 in which it appears (with the exception of the values of the `xrd:Query`, `xrd:Expires`, and
1831 `xrd:XRD/xrd:LocalID` elements). Resolution of a polyarchical CanonicalID **MUST**, like a Ref,
1832 return a different XRD from a different parent authority.

1833 A CanonicalID synonym **SHOULD NOT** be trusted unless it is verified. See section 12 for the
1834 rules governing verification of the binding between a CanonicalID and the QXRI or any other
1835 synonym in an XRD. A parent authority **SHOULD NOT** permit a child authority to edit the
1836 CanonicalID value in an XRD without authenticating the child authority and verifying that the child
1837 authority is authorized to use this CanonicalID value.

1838

12 Synonym Verification

1839
1840
1841
1842

For security purposes it is vital that applications be able to verify the binding between the identifier resolved to an XRDS document and any synonyms asserted in the final XRD. This applies in particular to CanonicalID synonyms because they are the identifiers applications should store and use to persistently reference a resource.

1843
1844
1845
1846

Applications MAY perform their own synonym verification, or they MAY request an XRI resolver perform this function by using the Resolution Output Format media type parameter `cid`. The following synonym verification tests MUST be applied by an XRI resolver if `cid=true`, and MUST NOT be applied if `cid=false` or the parameter is absent or empty.

1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857

1. If the value of the `xrd:XRD/xrd:CanonicalID` element is a hierarchical synonym as defined in section 11.1, it MUST be verified as specified in section 12.1.
2. If the value of the `xrd:XRD/xrd:CanonicalID` element is a polyarchical synonym as defined in section 11.1, it MUST be verified as specified in section 12.2.
3. In all cases, because synonym verification depends on trusting each authority in the resolution chain, trusted resolution (section 8) SHOULD be used with either `https=true` and/or `saml=true` to provide additional assurance of the authenticity of the results..
4. In all cases, if service reference processing as defined in section 13.2 is necessary, the values of ALL synonym elements in the XRD returned from resolving the service Ref MUST exactly match the values of ALL synonym elements in the XRD containing the service Ref.

1858
1859
1860
1861

A resolver MUST NOT return `<Status code="100">SUCCESS</Status>` unless all tests above are successful. If any verification test fails, resolution MAY continue, but if successful the resolver MUST return `<Status code="102">CID_NOT_VERIFIED</Status>` in the final XRD.

1862
1863
1864
1865
1866
1867
1868
1869
1870

IMPORTANT: There is no guarantee that all XRDs that describe the same child authority will return the same verified CanonicalID. Different parent authorities may assert different CanonicalIDs for the same child authority and all of these may all be verifiable. In addition, due to reference processing, the verified CanonicalID returned for an XRI MAY differ depending on the resolution input parameters. For example, as described in section 13.1.2, a request for a specific service endpoint type that is not found in the final XRD of the initial XRDS document may trigger reference processing resulting in a second nested XRDS document. The final XRD in the nested XRDS document may come from a different parent authority and have a different but still verifiable CanonicalID for the child authority.

1871

12.1 Hierarchical Verification Rules

1872

12.1.1 HTTP(S) URIs

1873
1874
1875

For HTTP(S) URIs, hierarchical synonym verification is supported only for fragments. By definition, an HTTP(S) URI that includes a fragment is a valid GlobalID synonym and a valid hierarchical CanonicalID synonym for the same HTTP(S) URI without the fragment.

1876

12.1.2 XRIs

1877
1878

To verify that an XRI is a valid GlobalID synonym or a valid hierarchical CanonicalID synonym for a QXRI, all the following tests MUST be successful.

- 1879 1. The value of the `xrd:XRD/xrd:ProviderID` element in the XRD from the community
1880 root authority MUST match the value expected from the configuration of the resolver for
1881 that community root authority.
- 1882 2. The value of the `xrd:XRD/xrd:ProviderID` element in any subsequent XRD MUST
1883 match the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in the
1884 authority resolution service endpoint of its parent XRD.
- 1885 3. The value of each subsegment of the GlobalID or CanonicalID MUST match either the
1886 Query or a valid LocalID in each corresponding XRD in the resolution chain.

1887 In other words, the set of all possible XRIs that are valid GlobalID synonyms or valid hierarchical
1888 CanonicalID synonyms for a QXRI can be constructed hierarchically from the sets of all valid
1889 LocalID synonyms asserted in each leg of the QXRI resolution chain.

1890 12.2 Polyarchical Verification Rules

1891 12.2.1 HTTP(S) URIs

1892 To verify that a second HTTP(S) URI is a valid polyarchical CanonicalID synonym for a query
1893 HTTP(S) URI, the second HTTP(S) URI MUST resolve successfully as specified in section 5 and
1894 the resulting XRD MUST contain a Backref whose value matches the query HTTP(S) URI.

1895 12.2.2 XRIs

1896 To verify that an XRI is a valid polyarchical CanonicalID synonym for a QXRI, the XRI MUST
1897 resolve successfully to a separate XRD containing a Backref whose value matches either the
1898 original QXRI or a verified GlobalID synonym of the original QXRI as specified in section 12.1.

1899 Note that the resulting XRD MAY also contain a Ref to the original QXRI or a verified GlobalID
1900 synonym of the original QXRI. In this case the authority references are circular.

1901 12.3 Hierarchical Verification Examples

1902 12.3.1 HTTP(S) URI to HTTP(S) URI

- 1903 • Query: `http://example.com/user`
- 1904 • CanonicalID: `http://example.com/user#1234`

1905 First XRDS (simplified for illustration purposes):

```
1906 <XRDS>  
1907 <XRD>  
1908 <CanonicalID>http://example.com/user#1234</CanonicalID>  
1909 <Service priority="10">  
1910 ...  
1911 </Service>  
1912 ...  
1913 </XRD>  
1914 </XRDS>
```

1915 No further resolution is necessary because this CanonicalID verifies by the HTTP(S) URI
1916 fragment rule in section 12.1.1.

1917 12.3.2 XRI to XRI

- 1918 • Query: `=example.name*delegate.name`
- 1919 • CanonicalID: `!=1000.62b1.44fd.2855!1234`

1920 First XRDS (for =example.name*delegate.name):

```
1921 <XRDS>
1922 <XRD>
1923 <Query>*example.name</Query>
1924 <ProviderID>xri://=</ProviderID>
1925 <LocalID>!1000.62b1.44fd.2855</LocalID>
1926 <GlobalID>=!1000.62b1.44fd.2855</GlobalID>
1927 <CanonicalID>=!1000.62b1.44fd.2855</CanonicalID>
1928 <Service>
1929 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
1930 <Type>xri://$res*auth*($v*2.0)</Type>
1931 <MediaType>application/xrds+xml</MediaType>
1932 <URI priority="10">http://resolve.example.com</URI>
1933 <URI priority="15">http://resolve2.example.com</URI>
1934 <URI>https://resolve.example.com</URI>
1935 </Service>
1936 ...
1937 </XRD>
1938 <XRD>
1939 <Query>*delegate.name</Query>
1940 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
1941 <LocalID>!1234</LocalID>
1942 <GlobalID>=!1000.62b1.44fd.2855!1234</GlobalID>
1943 <CanonicalID>=!1000.62b1.44fd.2855!1234</CanonicalID>
1944 <Service priority="1">
1945 ...
1946 </Service>
1947 ...
1948 </XRD>
1949 </XRDS>
```

1950 No further resolution is necessary because this CanonicalID satisfies the hierachical XRI
1951 synonym rules in section 12.1.2:

- 1952 • The ProviderID in the first XRD matches what is configured in the resolver (not visible in the
1953 example itself; only the resolver can confirm this).
- 1954 • The ProviderID in the second XRD matches the ProviderID given in the authority resolution
1955 service endpoint in the first XRD.
- 1956 • Each subsegment of the CanonicalID matches a valid LocalID in each XRD in the resolution
1957 chain.

1958 12.4 Polyarchical Verification Examples

1959 12.4.1 HTTP(S) URI to HTTP(S) URI

- 1960 • Query: http://example.com/user
- 1961 • CanonicalID: https://different.example.net/path/user

1962 First XRDS (for http://example.com/user):

```
1963 <XRDS>
1964 <XRD>
1965 <CanonicalID>https://different.example.net/path/user</CanonicalID>
1966 <Service priority="10">
1967 ...
1968 </Service>
1969 ...
1970 </XRD>
1971 </XRDS>
```

1972 Second XRDS (for `https://different.example.net/path/user`):

```
1973 <XRDS>
1974 <XRD>
1975 <CanonicalID>https://different.example.net/path/user</CanonicalID>
1976 <Backref>http://example.com/user</Backref>
1977 <Service priority="10">
1978 ...
1979 </Service>
1980 ...
1981 </XRD>
1982 </XRDS>
```

1983 12.4.2 HTTP(S) URI to XRI

- 1984 • Query: `http://example.com/user`
- 1985 • CanonicalID: `!=1000.62b1.44fd.2855`

1986 First XRDS (for `http://example.com/user`):

```
1987 <XRDS>
1988 <XRD>
1989 <CanonicalID>!=1000.62b1.44fd.2855</CanonicalID>
1990 <Service priority="10">
1991 ...
1992 </Service>
1993 ...
1994 </XRD>
1995 </XRDS>
```

1996 Second XRDS (for `!=1000.62b1.44fd.2855`):

```
1997 <XRDS>
1998 <XRD>
1999 <Query>!1000.62b1.44fd.2855</Query>
2000 <ProviderID>xri://=</ProviderID>
2001 <GlobalID>!=1000.62b1.44fd.2855</GlobalID>
2002 <CanonicalID>!=1000.62b1.44fd.2855</CanonicalID>
2003 <Backref>http://example.com/user</Backref>
2004 <Service priority="10">
2005 ...
2006 </Service>
2007 ...
2008 </XRD>
2009 </XRDS>
```

2010 12.4.3 XRI to HTTP(S) URI

- 2011 • Query: `=example.name`
- 2012 • CanonicalID: `https://example.com/user`

2013 First XRDS (for `=example.name`):

```
2014 <XRDS>
2015 <XRD>
2016 <Query>*example.name</Query>
2017 <ProviderID>xri://=</ProviderID>
2018 <LocalID>!1000.62b1.44fd.2855</LocalID>
2019 <GlobalID>!=1000.62b1.44fd.2855</GlobalID>
2020 <CanonicalID>https://example.user</CanonicalID>
2021 <Service priority="10">
2022 ...
```

2023
2024
2025
2026

```
</Service>
...
</XRD>
</XRDS>
```

2027 Second XRDS (for `https://example.com/user`):

2028
2029
2030
2031
2032
2033
2034
2035
2036
2037

```
<XRDS>
  <XRD>
    <CanonicalID>https://example.com/user</CanonicalID>
    <Backref>=!1000.62b1.44fd.2855</Backref>
    <Service priority="10">
      ...
    </Service>
  </XRD>
</XRDS>
```

2038 Note that the Backref is not to the original QXRI, `=example.name`, because it is a reassignable
2039 identifier. It is more secure to use a Backref to `=!f831.62b1.44fd.2855` because it is a
2040 persistent identifier that the parent authority `xri://=` asserts will not be reassigned (by using
2041 XRI ! syntax).

2042 12.4.4 XRI to XRI

- 2043 • Query: `=example.name*delegate.name`
- 2044 • CanonicalID: `@!1000.f3da.9056.aca3!5555`

2045 First XRDS (for `=example.name*delegate.name`):

2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074

```
<XRDS>
  <XRD>
    <Query>*example.name</Query>
    <ProviderID>xri://=</ProviderID>
    <LocalID>!1000.62b1.44fd.2855</LocalID>
    <GlobalID>=!1000.62b1.44fd.2855</GlobalID>
    <CanonicalID>=!1000.62b1.44fd.2855</CanonicalID>
    <Service>
      <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <MediaType>application/xrds+xml</MediaType>
      <URI priority="10">http://resolve.example.com</URI>
      <URI priority="15">http://resolve2.example.com</URI>
      <URI>https://resolve.example.com</URI>
    </Service>
  </XRD>
  <XRD>
    <Query>*delegate.name</Query>
    <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
    <LocalID>!1234</LocalID>
    <GlobalID>=!1000.62b1.44fd.2855!1234</GlobalID>
    <CanonicalID>@!1000.f3da.9056.aca3!5555</CanonicalID>
    <Service priority="1">
      ...
    </Service>
  </XRD>
</XRDS>
```

- 2075 • Second XRDS (for `@!1000.f3da.9056.aca3!5555`):

```

2076 <XRDS>
2077 <XRD>
2078 <Query>!1000.f3da.9056.aca3</Query>
2079 <ProviderID>xri://@</ProviderID>
2080 <GlobalID>!1000.f3da.9056.aca3</GlobalID>
2081 <CanonicalID>!1000.f3da.9056.aca3</CanonicalID>
2082 <Service>
2083 <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
2084 <Type>xri://$res*auth*($v*2.0)</Type>
2085 <MediaType>application/xrds+xml</MediaType>
2086 <URI priority="10">http://resolve.example.com</URI>
2087 <URI priority="15">http://resolve2.example.com</URI>
2088 <URI>https://resolve.example.com</URI>
2089 </Service>
2090 ...
2091 </XRD>
2092 <XRD>
2093 <Query>!5555</Query>
2094 <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
2095 <CanonicalID>!1000.f3da.9056.aca3!5555</CanonicalID>
2096 <Backref>=!1000.62b1.44fd.2855!1234</Backref>
2097 <Service priority="1">
2098 ...
2099 </Service>
2100 ...
2101 </XRD>
2102 </XRDS>

```

2103 Note again that the Backref is not to the original QXRI, =example.name*delegate.name, but
2104 to the more secure persistent identifier =!f831.62b1.44fd.2855!1234.

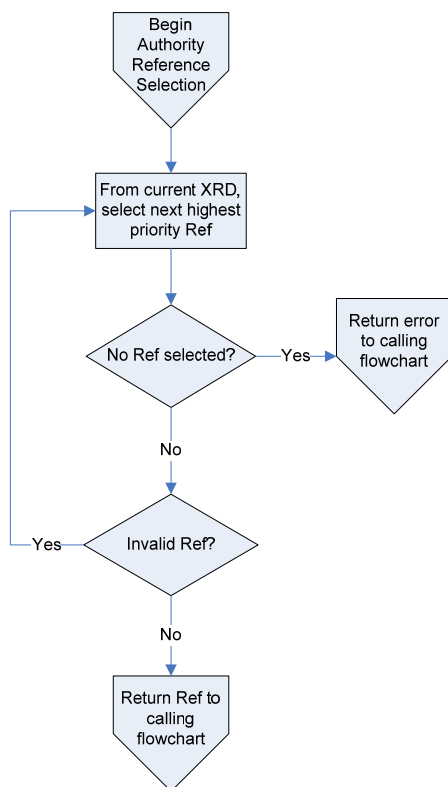
2105 13 Reference Processing

2106 13.1 Authority References

2107 This section contains the normative rules for processing of authority references in XRI resolution.

2108 13.1.1 Processing Rules

2109 Figure 11 is an overview of the logical flow of selecting an authority reference for processing.



2110

2111 *Figure 11: Flowchart for selecting an authority reference for processing.*

2112 Following are the normative rules that apply to authority reference processing:

- 2113 1. Authority reference processing will only be performed if the `refs` media type parameter
2114 (Table 6) is omitted or its value is `true`. If the value is `false` and the XRD contains at
2115 least one `xrd:Ref` element that could be followed, the resolver MUST return a response
2116 with a status code of 101 `REF_NOT_FOLLOWED`.
- 2117 2. The resolver MUST begin by selecting the highest priority `xrd:XRD/xrd:Ref` element in
2118 the current XRD. If not found, the result is an error and the resolver MUST proceed as
2119 defined in section 7 for authority resolution or section 10 for service endpoint selection.

- 2120 3. The contents of the selected `xrd:XRD/xrd:Ref` element MUST be a valid HTTP(S) URI
2121 or a valid QXRI as defined in section 6.1.1. If not, it MUST be ignored and step 1
2122 repeated to select the next highest priority reference.
- 2123 4. If the value is an HTTP(S) URI and the original XRI resolution parameter specified that
2124 `https=true`, then the value MUST be an HTTPS URI. If not, it is an invalid reference and
2125 MUST be ignored and step 1 repeated to select the next highest priority reference.
- 2126 5. If the value of the selected `xrd:XRD/xrd:Ref` element is an HTTP(S) URI, the resolver
2127 MUST resolve it to a new XRDS document as specified in section 5.
- 2128 6. If the value of the selected `xrd:XRD/xrd:Ref` element is a QXRI, the resolver MUST
2129 begin resolution of a new XRDS document beginning with the community root authority of
2130 the authority reference XRI as defined in section 7.1.3. The resolver MUST use the same
2131 resolution input parameters as for the original QXRI. For reference processing to
2132 complete successfully, the resolver MUST complete resolution of the entire Authority
2133 String of the reference XRI (including following any further authority references if
2134 necessary). If the reference XRI includes a Path String or Query String (including any
2135 resolution query parameters), they MUST be ignored.
- 2136 7. If authority reference processing is successful and the Resolution Output Format is an
2137 XRDS document, the XRDS document resulting from authority reference processing
2138 MUST be nested inside the parent XRDS document as defined in section 13.1.2.
- 2139 8. If authority reference processing is successful and the Resolution Output Format is an
2140 XRD document, the output MUST be the final XRD document returned.
- 2141 9. If authority reference processing is successful and the Resolution Output Format is a URI
2142 List or an HTTP(S) redirect, it MUST be based on the final XRD document returned.
- 2143 10. If authority reference processing fails, the resolver's work is complete and it MUST return
2144 an error as defined in section 14. Since authority reference processing is another iteration
2145 of authority resolution, it will typically be an authority resolution error.

2146 13.1.2 Nesting XRDS Documents

2147 If authority reference processing is successful, it will produce a new XRDS document that
2148 describes the reference. If the final requested Resolution Output format is NOT an XRDS
2149 document, this XRDS document is only needed to obtain the metadata necessary to continue
2150 resolution. However, if the final requested Resolution Output Format is an XRDS document, this
2151 new XRDS document MUST be included in the containing XRDS document immediately following
2152 the `xrd:XRDS` element that contains the `xrd:Ref` element being followed. In addition, the
2153 `xrds:XRDS/@xrds:ref` attribute of this nested XRDS document MUST be set to the exact
2154 value of the `xrd:XRDS/xrd:Ref` element it describes.

2155 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
2156 original QXRI even if resolution traverses authority references. Note that nested XRDS
2157 documents do not include an XRD for the community root subsegment because this is part of the
2158 configuration of the resolver.

2159 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD with an
2160 `xml:id` attribute value matching the `xml:id` attribute value of any previous XRD in the chain of
2161 resolution requests beginning with the original QXRI, the resolver MUST replace this XRD with an
2162 empty XRD element. The resolver MUST set this empty element's `xrd:idref` attribute value to
2163 the value of the `xrd:XRDS/xml:id` attribute of the matched XRD element. This prevents
2164 conflicting `xrd:XRDS/xml:id` values.

2165 In the following example the original query XRI is `xri://@a*b*c`. The XRD for `xri://@a*b`
2166 does not contain an authority resolution service endpoint but includes an authority reference to
2167 `xri://@x*y`. The elements and attributes specific to authority reference processing are shown
2168 in bold.

```

2169 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2170   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2171     <Query>*a</Query>
2172     ...
2173     <Service>
2174       <Type>xri://$res*auth*($v*2.0)</Type>
2175       <URI>http://a.example.com</URI>
2176     </Service>
2177   </XRD>
2178   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2179     <Query>*b</Query>
2180     ...
2181     <Ref>xri://@x*y</Ref>
2182     <Service>
2183       ...no authority resolution service endpoint...
2184     </Service>
2185   </XRD>
2186   <XRDS ref="xri://@x*y">
2187     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2188       <Query>*x</Query>
2189       ...
2190       <Service>
2191         <Type>xri://$res*auth*($v*2.0)</Type>
2192         <URI>http://x.example.com</URI>
2193       </Service>
2194     </XRD>
2195     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2196       <Query>*y</Query>
2197       ...
2198       <Service>
2199         <Type>xri://$res*auth*($v*2.0)</Type>
2200         <URI>http://y.example.com</URI>
2201       </Service>
2202     </XRD>
2203   </XRDS>
2204   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2205     <Query>*c</Query>
2206     ...
2207     <Service>
2208       ...final service endpoints described here...
2209     </Service>
2210   </XRD>
2211 </XRDS>

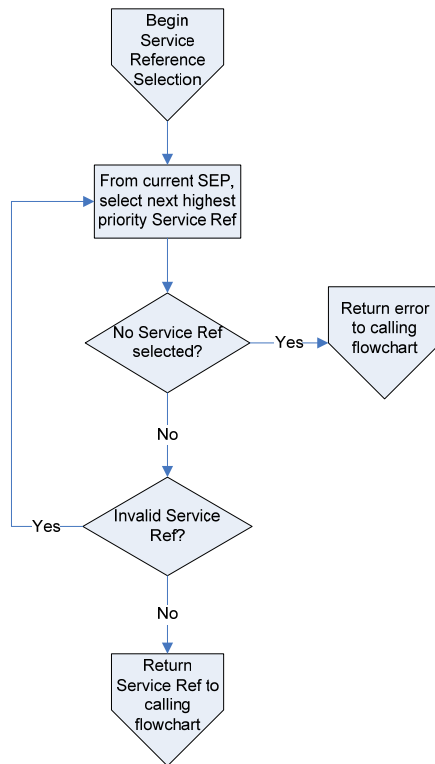
```

2212 13.2 Service References

2213 This section contains the normative rules for processing of service references in XRI resolution.

2214 13.2.1 Processing Rules

2215 Figure 11 is an overview of the logical flow of selecting a service reference for processing. The
2216 process is essentially identical to that of Figure 11 for authority reference processing.



2217

2218 *Figure 12: Flowchart for selecting a service reference for processing.*

2219 Following are the normative rules that apply to service reference processing:

- 2220 1. If service endpoint selection is requested and the highest priority service endpoint in the
 2221 final XRD DOES NOT contain an `xrd:XRD/xrd:Service/xrd:URI` element but
 2222 DOES contain at least one `xrd:XRD/xrd:Service/xrd:Ref` element, the resolver
 2223 MUST begin reference processing by selecting the highest priority
 2224 `xrd:XRD/xrd:Service/xrd:Ref` element in the XRD.
- 2225 2. The contents of the selected `xrd:XRD/xrd:Service/xrd:Ref` element MUST be a
 2226 valid HTTP(S) URI. If not, it MUST be ignored and step 1 repeated to select the next
 2227 highest priority reference.
- 2228 3. If the original XRI resolution parameters specified that `https=true`, then the contents of
 2229 the selected `xrd:XRD/xrd:Service/xrd:Ref` element MUST be a valid HTTPS URI.
 2230 If not, it is an invalid service reference and MUST be ignored and step 1 repeated to
 2231 select the next highest priority reference.
- 2232 4. Once a valid service reference has been selected, if the
 2233 `xrd:XRD/xrd:Service/xrd:Ref` element includes the `xrd:append` attribute, the
 2234 resolver MUST construct the final HTTP(S) URI as defined in section 10.7.
- 2235 5. The resolver MUST directly request a new XRDS document from the final HTTP(S) URI
 2236 as defined in section 7.1.2. For service processing to complete successfully, the resolver
 2237 MUST also complete service endpoint selection (including following any further service
 2238 references if necessary) using the same service endpoint selection parameters as the
 2239 original query.

- 2240 6. If service reference processing is successful and the Resolution Output Format is an
 2241 XRDS document, the XRD document resulting from service reference processing **MUST**
 2242 sequentially follow the XRD containing the service reference as defined in section 13.2.2
- 2243 7. If service reference processing is successful and the Resolution Output Format is an
 2244 XRD document, the output **MUST** be the final XRD document returned.
- 2245 8. If service reference processing is successful and the Resolution Output Format is a URI
 2246 List or an HTTP(S) redirect, it **MUST** be based on the final XRD document returned.

2247 13.2.2 Adding XRD Documents

2248 If a service reference is followed successfully, it will produce an XRDS document containing an
 2249 XRD element with a service endpoint satisfying the original service endpoint selection
 2250 parameters. This XRD element **MUST** be included in the containing XRDS document immediately
 2251 following the `xrd:XRD` element containing the service reference being followed. In addition, the
 2252 resolver **MUST** set the value of the `xrd:XRD/@xrd:ref` attribute of this XRD document to the
 2253 value of the `xrd:XRD/xrd:Service/xrd:Ref` element used to request it.

2254 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
 2255 original QXRI even if resolution traverses one or more service references.

2256 For SAML trusted resolution, two special rules apply to the XRD elements resulting from service
 2257 reference processing:

- 2258 1. If an XRD is returned with an `xml:id` attribute value matching the `xml:id` attribute
 2259 value of any previous XRD in the chain of resolution requests beginning with the original
 2260 QXRI, the resolver **MUST** replace this XRD with an empty XRD element. The resolver
 2261 **MUST** set this empty element's `xrd:idref` attribute value to the value of the
 2262 `xrd:XRD/xml:id` attribute of the matched XRD element. This prevents conflicting
 2263 `xrd:XRD/xml:id` values.
- 2264 2. The `xrd:XRD/@xrd:ref` attribute appended to the XRD by the resolver **MUST NOT** be
 2265 included in the SAML signature validation processing defined in section 8.2.4.

2266 In the following example the original query XRI is `xri://@a*b*c`. The XRD for `xri://@a*b`
 2267 contains an authority resolution service endpoint without a URI element but with a service
 2268 reference to `http://x.example.com/xri/`. This produces a subsequent XRD with the
 2269 requested service endpoint type and URI element. The elements and attributes specific to service
 2270 reference processing are shown in bold.

```

2271 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2272   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2273     <Query>*a</Query>
2274     ...
2275     <Service>
2276       <Type>xri://$res*auth*($v*2.0)</Type>
2277       <URI>http://a.example.com</URI>
2278     </Service>
2279   </XRD>
2280   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2281     <Query>*b</Query>
2282     ...
2283     <Service>
2284       <Type>xri://$res*auth*($v*2.0)</Type>
2285       ...no URI element...
2286       <Ref append="qxri">http://x.example.com/xri/</Ref>
2287     </Service>
2288   </XRD>
2289   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0"
2290     ref="http://x.example.com/xri/@a*b*c">
2291     <Service>
  
```

```
2292         <Type>xri://$res*auth*($v*2.0)</Type>
2293         <URI>http://b.example.com</URI>
2294     </Service>
2295 </XRD>
2296 </XRDS>
2297 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2298     <Query>*c</Query>
2299     ...
2300     <Service>
2301         ...final service endpoints described here...
2302     </Service>
2303 </XRD>
2304 </XRDS>
```

2305

14 Error Processing

2306

14.1 Error Codes

2307 XRI resolution error codes are patterned after the HTTP model. They are broken into three major
2308 categories:

- 2309
- 1xx: Success—the requested resolution operation was completed successfully.
- 2310
- 2xx: Permanent errors—the resolver encountered an error from which it could not recover.
- 2311
- 3xx: Temporary errors—the resolver encountered an error condition that may be only
- 2312 temporary.

2313 Each major category is broken into five minor categories:

- 2314
- x0x: General error that may take place during any phase of resolution.
- 2315
- x1x: Input error.
- 2316
- x2x: Generic authority resolution error.
- 2317
- x3x: Trusted authority resolution error.
- 2318
- x4x: Service endpoint (SEP) selection error.

2319 The full list of XRI resolution error codes is defined in Table 24.

2320

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.
102	CID_NOT_VERIFIED	Any	Resolution was successful but CanonicalID verification failed – see section 12.
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 Output Format 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.
224	INACTIVE	Authority resolution	The query XRI has been assigned but the authority does not provide resolution metadata.
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).
--	--	--	---

2321 *Table 24: Error codes for XRI resolution.*

2322 As defined in section 6.2, when resolution output is an error and the output format is an XRDS
 2323 Document or XRD Document, the error code is returned as the value of the `xrd:code` attribute
 2324 of the `xrd:Status` element. When the resolution output is a URI List, the error code is returned
 2325 as the first line of a plain text message.

2326 14.2 Error Context Strings

2327 Each error code in Table 24 MAY be returned with an optional error context string that provides
 2328 additional human-readable information about the error. When resolution output is an XRDS
 2329 Document or XRD Document, this string is returned as the contents of the `xrd:Status` element.
 2330 When the resolution output is a URI List, this string MUST be returned as the second line of a
 2331 plain text message as specified in section 9.6. Implementers SHOULD provide error context
 2332 strings with additional information about an error and possible solutions whenever it can be
 2333 helpful to developers or end users.

2334 14.3 Error Handling in Recursing and Proxy Resolution

2335 In recursing and proxy resolution (sections 7.1.6 and 9), a server is acting as a client resolver for
 2336 other authority resolution service endpoints. If in this intermediary capacity it receives an
 2337 unrecoverable error, it MUST return the error to the originating client in the output format
 2338 specified by the value of the requested Resolution Output Format as defined in section 6.2.

2339 If the output format is an XRDS Document, it MUST contain `xrd:XRD` elements for all
 2340 subsegments successfully resolved or retrieved from cache prior to the error. The final `xrd:XRD`
 2341 element MUST include the `xrd:Query` element that produced the error and the `xrd:Status`
 2342 element that describes the error as defined above.

2343 If the output format is an XRD Document, it MUST include the `xrd:Query` element that produced
 2344 the error and the `xrd:Status` element that describes the error as defined above.

2345 If this output format is a URI List, it MUST be returned with the content type `text/plain`. The
 2346 first line MUST consist of only the numeric error code as defined in section 14.1 followed by a
 2347 CRLF. The second line is OPTIONAL; if present it MUST be the error context string as defined in
 2348 section 14.2.

2349 If the value of the Resolution Output Format is null (which can only happen in proxy resolution as
 2350 described in section 9.5), rather than returning an HTTP(S) redirect, a proxy resolver SHOULD
 2351 return a human-readable error message with a media type of either `text/plain` or `text/html`.
 2352 It is particularly important in this case to return an error message that will be understandable to an
 2353 end-user who may have no understanding of XRI resolution or the fact that the error is coming
 2354 from an XRI proxy resolver.

2355 15 Use of HTTP(S)

2356 15.1 HTTP Errors

2357 When a resolver encounters fatal HTTP(S) errors during the resolution process, it **MUST** return
2358 the appropriate XRI resolution error code and error message as defined in section 0. In this way
2359 calling applications do not have to deal separately with XRI and HTTP error messages.

2360 15.2 HTTP Headers

2361 15.2.1 Caching

2362 The HTTP caching capabilities described by **[RFC2616]** should be leveraged for all types of XRI
2363 resolution service. Specifically, implementations **SHOULD** implement the caching model
2364 described in section 13 of **[RFC2616]**, and in particular, the “Expiration Model” of section 13.2, as
2365 this requires the fewest round-trip network connections.

2366 All XRI resolution servers **SHOULD** send the Cache-Control or Expires headers in their
2367 responses per section 13.2 of **[RFC2616]** unless there are overriding security or policy reasons to
2368 omit them.

2369 Note that HTTP Cache headers **SHOULD NOT** conflict with expiration information in an XRD.
2370 That is, the expiration date specified by HTTP caching headers **SHOULD NOT** be later than any
2371 of the expiration dates for any of the `xrd:Expires` elements returned in the HTTP response.
2372 This implies that recursing and proxy resolvers **SHOULD** compute the “soonest” expiration date
2373 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching
2374 headers for the HTTP response.

2375 15.2.2 Location

2376 During HTTP interaction, “Location” headers may be present per **[RFC2616]** (i.e., during 3XX
2377 redirects). Redirects **SHOULD** be made cacheable through appropriate HTTP headers, as
2378 specified in section 15.2.1.

2379 15.2.3 Content-Type

2380 For authority resolution, the “Content-type” header in the 2XX responses **MUST** contain the
2381 media type identifier values specified in Table 10 (for generic resolution), Table 14 (for HTTPS
2382 trusted resolution), Table 15 (for SAML trusted resolution), or Table 16 (or HTTPS+SAML trusted
2383 resolution).

2384 Following service endpoint selection, clients and servers **MAY** negotiate content type using
2385 standard HTTP content negotiation features. Regardless of whether this feature is used,
2386 however, the server **MUST** respond with an appropriate media type in the “Content-type” header
2387 if the resource is found and an appropriate content type is returned.

2388 15.3 Other HTTP Features

2389 HTTP provides a number of other features including transfer-coding, proxying, validation-model
2390 caching, and so forth. All these features may be used insofar as they do not conflict with the
2391 required uses of HTTP described in this document.

2392 15.4 Caching and Efficiency

2393 15.4.1 Resolver Caching

2394 In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the
2395 application level. For best results, however, resolution clients SHOULD be conservative with
2396 caching expiration semantics, including cache expiration dates. This implies that in a series of
2397 HTTP redirects, for example, the results of the entire process SHOULD only be cached as long
2398 as the shortest period of time allowed by any of the intermediate HTTP responses.

2399 Because not all HTTP client libraries expose caching expiration to applications, identifier
2400 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the
2401 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments
2402 should be mindful of limitations in current HTTP clients and proxies.

2403 The cache expiration time of an XRD may also be explicitly limited by the parent authority. If the
2404 expiration time in the `xrd:Expires` element is sooner than the expiration time calculated from
2405 the HTTP caching semantics, the XRD MUST be discarded before the expiration time in
2406 `xrd:Expires`. Note also that a `saml:Assertion` element returned during SAML trusted
2407 resolution has its own signature expiration semantics as defined in [SAML]. While this may
2408 invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if
2409 it is not expired by HTTP caching semantics or the `xrd:Expires` element.

2410 With both application-level and HTTP-level caching, the resolution process is designed to have
2411 minimal overhead. Resolution of each qualified subsegment of an XRI authority segment is a
2412 separate step described by a separate XRD, so intermediate results can typically be cached in
2413 their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified
2414 subsegments, which are common to more identifiers, will naturally result in a greater number of
2415 cache hits than resolution of lower-level subsegments.

2416 15.4.2 Synonyms

2417 The publication of synonyms in XRD documents can further increase cache efficiency. If an XRI
2418 resolution request produces a cache hit on a synonym, the following rules apply:

- 2419 1. If the cache hit is on a LocalID synonym, the resolver MAY return the cached XRD
2420 document if: a) it is from the correct ProviderID, b) it has not expired, and c) it was
2421 obtained using the same trusted resolution and synonym verification parameters as the
2422 current resolution request.
- 2423 2. If the cache hit is on a GlobalID synonym, the resolver MAY return the entire cached
2424 XRDS document if: a) it has not expired, and b) it was obtained using the same trusted
2425 resolution and synonym verification parameters as the current resolution request.

2426 IMPORTANT: The effect of these rules is that the application calling an XRI resolver MAY receive
2427 back an XRD document, or an XRDS document containing XRD document(s), in which the
2428 `<Query>` element does not match the resolution request, but in which a `<LocalID>` element
2429 match does match the resolution request.

2430 16 Extensibility and Versioning

2431 16.1 Extensibility

2432 16.1.1 Extensibility of XRDs

2433 The XRD schema in Appendix A use an an open-content model that is designed to be extended
2434 with other metadata. In most places, extension elements and attributes from namespaces other
2435 than `xri://$xrd*($v*2.0)` are explicitly allowed. These extension points are designed to
2436 simplify default processing using a “Must Ignore” rule. The base rule is that unrecognized
2437 elements and attributes, and the content and child elements of unrecognized elements, MUST be
2438 ignored. As a consequence, elements that would normally be recognized by a processor MUST
2439 be ignored if they appear as descendants of an unrecognized element.

2440 Extension elements MUST NOT require new interpretation of elements defined in this document.
2441 If an extension element is present, a processor MUST be able to ignore it and still correctly
2442 process the XRD document.

2443 Extension specifications MAY simulate “Must Understand” behavior by applying an “enclosure”
2444 pattern. Elements defined by the XRD schema in Appendix A whose meaning or interpretation is
2445 modified by extension elements can be wrapped in a extension container element defined by the
2446 extension specification. This extension container element SHOULD be in the same namespace
2447 as the other extension elements defined by the extension specification.

2448 Using this design, all elements whose interpretations are modified by the extension will now be
2449 contained in the extension container element and thus will be ignored by clients or other
2450 applications unable to process the extension. The following example illustrates this pattern using
2451 an extension container element from an extension namespace (`other:SuperService`) that
2452 contains an extension element (`other:ExtensionElement`):

```
2453 <XRD>  
2454   <Service>  
2455     ...  
2456   </Service>  
2457   <other:SuperService>  
2458     <Service>  
2459       ...  
2460       <other:ExtensionElement>...</other:ExtensionElement>  
2461     </Service>  
2462   </other:SuperService>  
2463 </XRD>
```

2464 In this example, the `other:ExtensionElement` modifies the interpretation or processing rules
2465 for the parent `xrd:Service` element and therefore must be understood by the consumer for the
2466 proper interpretation of the parent `xrd:Service` element. To preserve the correct interpretation
2467 of the `xrd:Service` element in this context, the `xrd:Service` element is “wrapped” so only
2468 consumers that understand elements in the `other:SuperService` namespace will attempt to
2469 process the `xrd:ProviderID` element.

2470 The addition of extension elements does not change the requirement for SAML signatures to be
2471 verified across all elements, whether recognized or not.

2472 16.1.2 Other Points of Extensibility

2473 The use of HTTP, XML, XRI, and URIs in the design of XRDS documents, XRD elements, and
2474 XRI resolution architecture provides additional specific points of extensibility:

- 2475 • Specification of new resolution service types or other service types using XRI or URIs as
2476 values of the `xrd:Type` element.
- 2477 • Specification of new resolution output formats or features using media types and media type
2478 parameters as values of the `xrd:MediaType` element as defined in [RFC2045] and
2479 [RFC2046].
- 2480 • HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- 2481 • Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- 2482 • Use of cross-references within XRIs, particularly for associating new types of metadata with a
2483 resource. See [XRISyntax] and [XRIMetadata].

2484 16.2 Versioning

2485 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,
2486 this section describes versioning guidelines.

2487 16.2.1 Version Numbering

2488 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number
2489 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version
2490 number *MajorA.MinorA* if and only if:

2491 $Major_B > Major_A$ OR (($Major_B = Major_A$) AND $Minor_B > Minor_A$)

2492 16.2.2 Versioning of the XRI Resolution Specification

2493 New releases of the XRI Resolution specification may specify changes to the resolution protocols
2494 and/or the XRD schema in Appendix A. When changes affect either of these, the resolution
2495 service type version number will be changed. Where changes are purely editorial, the version
2496 number will not be changed.

2497 In general, if a change is backward-compatible, the new version will be identified using the
2498 current major version number and a new minor version number. If the change is not backward-
2499 compatible, the new version will be identified with a new major version number.

2500 16.2.3 Versioning of XRDs

2501 The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have
2502 no specific knowledge of the elements it may contain. Therefore it has no version element, and
2503 can remain stable indefinitely because there is no need to version its namespace.

2504 The `xrd:XRD` element has an optional `xrd:version` attribute. When used, the value of this
2505 attribute MUST be the exact numeric version value of the XRI Resolution specification to which its
2506 containing elements conform.

2507 When new versions of the XRI Resolution specification are released, the namespace for the XRD
2508 schema may or may not be changed. If there is a major version number change, the namespace
2509 for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the
2510 namespace for the `xrd:XRD` schema may remain unchanged.

2511 With regard to versioning, this specification follows the same guidelines as established in section
2512 4.2.1 of [SAML]:

2513 *In general, maintaining namespace stability while adding or changing the content of a*
2514 *schema are competing goals. While certain design strategies can facilitate such changes,*
2515 *it is complex to predict how older implementations will react to any given change, making*
2516 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*
2517 *minor revisions is reserved, in the interest of namespace stability. Except in special*

2518 *circumstances (for example, to correct major deficiencies or to fix errors),*
2519 *implementations should expect forward-compatible schema changes in minor revisions,*
2520 *allowing new messages to validate against older schemas.*

2521 *Implementations SHOULD expect and be prepared to deal with new extensions and*
2522 *message types in accordance with the processing rules laid out for those types. Minor*
2523 *revisions MAY introduce new types that leverage the extension facilities described in [this*
2524 *section]. Older implementations SHOULD reject such extensions gracefully when they*
2525 *are encountered in contexts that dictate mandatory semantics.*

2526 **16.2.4 Versioning of Protocols**

2527 The protocols defined in this document may also be versioned by future releases of the XRI
2528 Resolution specification. If these protocols are not backward-compatible with older
2529 implementations, they will be assigned a new XRI with a new version identifier for use in
2530 identifying their service type in XRDs. See section 3.1.2.

2531 Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP
2532 provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an
2533 XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely
2534 to continue to use the same XRI to identify the protocol as was used in previous versions of the
2535 XRI Resolution specification.

2536

17 Security and Data Protection

2537
2538
2539
2540

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

2541

17.1 DNS Spoofing or Poisoning

2542
2543
2544
2545
2546
2547
2548
2549
2550

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

2551

17.2 HTTP Security

2552
2553
2554
2555

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

2556
2557
2558
2559

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

2560
2561

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

2562

17.3 SAML Considerations

2563
2564
2565

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

2566

17.4 Limitations of Trusted Resolution

2567
2568
2569
2570
2571

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

2572
2573
2574

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

2575 **17.5 Community Root Authorities**

2576 The XRI authority information for a community root needs to be well-known to the clients that
2577 request resolution within that community. For trusted resolution, this includes the authority
2578 resolution service endpoint URIs, the `xrd:XRD/xrd:ProviderID`, and the `ds:KeyInfo`
2579 information. An acceptable means of providing this information is for the community root authority
2580 to produce a self-signed XRD and publish it to a server-authenticated HTTPS endpoint. Special
2581 care should be taken to ensure the correctness of such an XRD; if this information is incorrect, an
2582 attacker may be able to convince a client of an incorrect result during trusted resolution.

2583 **17.6 Caching Authorities**

2584 In addition to traditional HTTP caching proxies, XRI proxy resolvers may be a part of the
2585 resolution topology. Such proxy resolvers should take special precautions against cache
2586 poisoning, as these caching entities may represent trusted decision points within a deployment's
2587 resolution architecture.

2588 **17.7 Recursing and Proxy Resolution**

2589 During recursing resolution, subsegments of the XRI authority segment for which the resolving
2590 network endpoint is not authoritative may be revealed to that service endpoint. During proxy
2591 resolution, some or all of an XRI is provided to the proxy resolver.

2592 In both cases, privacy considerations should be evaluated before disclosing such information.

2593 **17.8 Denial-Of-Service Attacks**

2594 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks
2595 typical of systems relying on DNS and HTTP.

A. Acknowledgments

2597 The editors would like to acknowledge the contributions of the OASIS XRI Technical Committee,
 2598 whose voting members at the time of publication were:

Comment [DSR20]: TO DO -
Needs updating.

- 2599 • Geoffrey Strongin, Advanced Micro Devices
- 2600 • Ajay Madhok, AmSoft Systems
- 2601 • Jean-Jacques Dubray, Attachmate
- 2602 • William Barnhill, Booz Allen and Hamilton
- 2603 • Drummond Reed, Cordance Corporation
- 2604 • Marc Le Maitre, Cordance Corporation
- 2605 • Dave McAlpin, Epok
- 2606 • Loren West, Epok
- 2607 • Peter Davis, NeuStar
- 2608 • Masaki Nishitani, Nomura Research
- 2609 • Nat Sakimura, Nomura Research
- 2610 • Tetsu Watanabe, Nomura Research
- 2611 • Owen Davis, PlaNetwork
- 2612 • Victor Grey, PlaNetwork
- 2613 • Fen Labalme, PlaNetwork
- 2614 • Mike Lindelsee, Visa International
- 2615 • Gabriel Wachob, Visa International
- 2616 • Dave Wentker, Visa International
- 2617 • Bill Washburn, XDI.ORG

2618 The editors also would like to acknowledge the following people for their contributions to previous
 2619 versions of the OASIS XRI specifications (affiliations listed for OASIS members):

- 2620 Thomas Bikeev, EAN International; Krishna Sankar, Cisco; Winston Bumpus, Dell; Joseph
 2621 Moeller, EDS; Steve Green, Epok; Lance Hood, Epok; Adarbad Master, Epok; Davis McPherson,
 2622 Epok; Chetan Sabnis, Epok; Phillipe LeBlanc, GemPlus; Jim Schreckengast, Gemplus; Xavier
 2623 Serret, Gemplus; John McGarvey, IBM; Reva Modi, Infosys; Krishnan Rajagopalan, Novell;
 2624 Tomonori Seki, NRI; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,
 2625 Verisign; Rajeev Maria, Visa International; Terence Spielman, Visa International; John Veizades,
 2626 Visa International; Lark Allen, Wave Systems; Michael Willett, Wave Systems; Matthew Dovey;
 2627 Eamonn Neylon; Mary Nishikawa; Lars Marius Garshol; Norman Paskin; Bernard Vatant.
- 2628 • A special acknowledgement to Jerry Kindall (Epok) for a full editorial review.

B. Non-Normative Text

2630

C. Revision History

2631 [optional; should not be included in OASIS Standards]

2632

Revision	Date	Editor	Changes Made
WD11 ED01	2007-05-23	Drummond Reed	All major changes from http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11 . A number of the more minor changes remain to be done.
WD11 ED02	2007-06-06	Drummond Reed	<p>Added content of Appendix F and G prepared by Gabe Wachob.</p> <p>Moved "Discovery of XRDS Documents from HTTP URIs" to section 4, added overview, added extensive feedback.</p> <p>Fixed bug in section 9, Pseudocode (for Service Endpoint Selection) spotted by Markus Sabadello.</p> <p>Numerous minor errata identified by Gabe Wachob and Marcus Sabadello fixed.</p> <p>Added comments to section 11, CanonicalID Verification, indicating work to be done.</p>
WD11 ED03	2007-07-24	Drummond Reed	<p>Added section 2, Conformance (still needs to be completed).</p> <p>Revised section 5.</p> <p>Added section 7.1.4.</p> <p>Added section 9.8.</p> <p>Added new section 11, Synonyms.</p> <p>Renamed and rewrote section 12, Synonym Verification.</p> <p>40+ other smaller changes as detailed on the XRI TC wiki at http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11.</p>

2633

2634

D. XML Schema for XRDS and XRD (Normative)

```
2635 <?xml version="1.0" encoding="UTF-8"?>
2636 <xs:schema targetNamespace="xri://$xrds" elementFormDefault="qualified"
2637 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds">
2638   <!-- Utility patterns -->
2639   <xs:attributeGroup name="otherattribute">
2640     <xs:anyAttribute namespace="##other" processContents="lax"/>
2641   </xs:attributeGroup>
2642   <xs:group name="otherelement">
2643     <xs:choice>
2644       <xs:any namespace="##other" processContents="lax"/>
2645       <xs:any namespace="##local" processContents="lax"/>
2646     </xs:choice>
2647   </xs:group>
2648   <!-- Patterns for elements -->
2649   <xs:element name="XRDS">
2650     <xs:complexType>
2651       <xs:sequence>
2652         <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2653       </xs:sequence>
2654       <xs:attributeGroup ref="xrds:otherattribute"/>
2655       <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
2656     </xs:complexType>
2657   </xs:element>
2658 </xs:schema>
2659
2660
2661 <?xml version="1.0" encoding="UTF-8"?>
2662 <xs:schema targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified"
2663 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2664 xmlns:xrd="xri://$xrd*($v*2.0)">
2665   <!-- Utility patterns -->
2666   <xs:attributeGroup name="otherattribute">
2667     <xs:anyAttribute namespace="##other" processContents="lax"/>
2668   </xs:attributeGroup>
2669   <xs:group name="otherelement">
2670     <xs:choice>
2671       <xs:any namespace="##other" processContents="lax"/>
2672       <xs:any namespace="##local" processContents="lax"/>
2673     </xs:choice>
2674   </xs:group>
2675   <xs:attributeGroup name="priorityAttrGrp">
2676     <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
2677   </xs:attributeGroup>
2678   <xs:attributeGroup name="selectionAttrGrp">
2679     <xs:attribute name="match" use="optional" default="default">
2680       <xs:simpleType>
2681         <xs:restriction base="xs:string">
2682           <xs:enumeration value="default"/>
2683           <xs:enumeration value="content"/>
2684           <xs:enumeration value="any"/>
2685           <xs:enumeration value="non-null"/>
2686           <xs:enumeration value="null"/>
2687           <xs:enumeration value="none"/>
2688         </xs:restriction>
2689       </xs:simpleType>
2690     </xs:attribute>
2691     <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
2692   </xs:attributeGroup>
2693   <xs:complexType name="URIPattern">
2694     <xs:simpleContent>
2695       <xs:extension base="xs:anyURI">
2696         <xs:attributeGroup ref="xrd:otherattribute"/>
2697       </xs:extension>
2698     </xs:simpleContent>
2699   </xs:complexType>
2700   <xs:complexType name="URIPriorityPattern">
```

```

2701     <xs:simpleContent>
2702         <xs:extension base="xrd:URIPattern">
2703             <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2704         </xs:extension>
2705     </xs:simpleContent>
2706 </xs:complexType>
2707 <xs:complexType name="StringPattern">
2708     <xs:simpleContent>
2709         <xs:extension base="xs:string">
2710             <xs:attributeGroup ref="xrd:otherattribute"/>
2711         </xs:extension>
2712     </xs:simpleContent>
2713 </xs:complexType>
2714 <xs:complexType name="StringSelectionPattern">
2715     <xs:simpleContent>
2716         <xs:extension base="xrd:StringPattern">
2717             <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
2718         </xs:extension>
2719     </xs:simpleContent>
2720 </xs:complexType>
2721 <!-- Patterns for elements -->
2722 <xs:element name="XRD">
2723     <xs:complexType>
2724         <xs:sequence>
2725             <xs:element ref="xrd:Query" minOccurs="0"/>
2726             <xs:element ref="xrd:Status" minOccurs="0"/>
2727             <xs:element ref="xrd:Expires" minOccurs="0"/>
2728             <xs:element ref="xrd:ProviderID" minOccurs="0"/>
2729             <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
2730             <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded"/>
2731             <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
2732             <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded"/>
2733             <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2734         </xs:sequence>
2735         <xs:attribute name="id" type="xs:ID"/>
2736         <xs:attribute name="idref" type="xs:IDREF" use="optional"/>
2737         <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0"/>
2738         <xs:attributeGroup ref="xrd:otherattribute"/>
2739     </xs:complexType>
2740 </xs:element>
2741 <xs:element name="Query" type="xrd:StringPattern"/>
2742 <xs:element name="Status">
2743     <xs:complexType>
2744         <xs:simpleContent>
2745             <xs:extension base="xrd:StringPattern">
2746                 <xs:attribute name="code" type="xs:int" use="required"/>
2747                 <xs:attributeGroup ref="xrd:otherattribute"/>
2748             </xs:extension>
2749         </xs:simpleContent>
2750     </xs:complexType>
2751 </xs:element>
2752 <xs:element name="Expires">
2753     <xs:complexType>
2754         <xs:simpleContent>
2755             <xs:extension base="xs:dateTime">
2756                 <xs:attributeGroup ref="xrd:otherattribute"/>
2757             </xs:extension>
2758         </xs:simpleContent>
2759     </xs:complexType>
2760 </xs:element>
2761 <xs:element name="ProviderID" type="xrd:URIPattern"/>
2762 <xs:element name="LocalID">
2763     <xs:complexType>
2764         <xs:simpleContent>
2765             <xs:extension base="xrd:StringPattern">
2766                 <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2767             </xs:extension>
2768         </xs:simpleContent>
2769     </xs:complexType>
2770 </xs:element>
2771 <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>

```

```

2772 <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
2773 <xs:element name="Service">
2774   <xs:complexType>
2775     <xs:sequence>
2776       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
2777       <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
2778       <xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
2779       <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
2780       <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
2781       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2782     </xs:sequence>
2783     <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2784     <xs:attributeGroup ref="xrd:otherattribute"/>
2785   </xs:complexType>
2786 </xs:element>
2787 <xs:element name="Type">
2788   <xs:complexType>
2789     <xs:simpleContent>
2790       <xs:extension base="xrd:URIPattern">
2791         <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
2792       </xs:extension>
2793     </xs:simpleContent>
2794   </xs:complexType>
2795 </xs:element>
2796 <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
2797 <xs:element name="Path" type="xrd:StringSelectionPattern"/>
2798 <xs:element name="URI">
2799   <xs:complexType>
2800     <xs:simpleContent>
2801       <xs:extension base="xrd:URIPattern">
2802         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2803         <xs:attribute name="append">
2804           <xs:simpleType>
2805             <xs:restriction base="xs:string">
2806               <xs:enumeration value="none"/>
2807               <xs:enumeration value="local"/>
2808               <xs:enumeration value="authority"/>
2809               <xs:enumeration value="path"/>
2810               <xs:enumeration value="query"/>
2811               <xs:enumeration value="qxri"/>
2812             </xs:restriction>
2813           </xs:simpleType>
2814         </xs:attribute>
2815       </xs:extension>
2816     </xs:simpleContent>
2817   </xs:complexType>
2818 </xs:element>
2819 </xs:schema>
2820
2821

```

2822 **E. RelaxNG Compact Syntax Schema for XRDS**
2823 **and XRD (Informative)**

2824 [TODO]

2825 **F. Media Type Definition for application/xrds+xml**
2826 **(Normative)**

2827 This section is prepared in anticipation of media type registration meeting the requirements of
2828 **[RFC4288]**.

2829 **Type name:** application

2830 **Subtype name:** xrds+xml

2831 **Required parameters:** None

2832 **Optional parameters:** See Table 6 of this document.

2833 **Encoding considerations:** Identical to those of "application/xml" as described in **[RFC3023]**,
2834 Section 3.2.

2835 **Security considerations:** As defined in this specification. In addition, as this media type uses the
2836 "+xml" convention, it shares the same security considerations as described in **[RFC3023]**,
2837 Section 10.

2838 **Interoperability considerations:** There are no known interoperability issues.

2839 **Published specification:** This specification.

2840 **Applications that use this media type:** Applications conforming to this specification use this
2841 media type.

2842 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
2843 Technical Committee Co-Chair, drummond.reed@cordance.net

2844 **Intended usage:** COMMON

2845 **Restrictions on usage:** None

2846 **Author:** OASIS XRI TC

2847 **Change controller:** OASIS XRI TC

2848
2849

G. Media Type Definition for application/xrd+xml (Normative)

2850 This section is prepared in anticipation of media type registration meeting the requirements of
2851 [RFC4288].

2852 **Type name:** application

2853 **Subtype name:** xrd+xml

2854 **Required parameters:** None

2855 **Optional parameters:** See Table 6 of this document.

2856 **Encoding considerations:** Identical to those of "application/xml" as described in [RFC3023],
2857 Section 3.2.

2858 **Security considerations:** As defined in this specification. In addition, as this media type uses the
2859 "+xml" convention, it shares the same security considerations as described in [RFC3023],
2860 Section 10.

2861 **Interoperability considerations:** There are no known interoperability issues.

2862 **Published specification:** This specification.

2863 **Applications that use this media type:** Applications conforming to this specification use this
2864 media type.

2865 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
2866 Technical Committee Co-Chair, drummond.reed@cordance.net

2867 **Intended usage:** COMMON

2868 **Restrictions on usage:** None

2869 **Author:** OASIS XRI TC

2870 **Change controller:** OASIS XRI TC

2871

H. Example Local Resolver Interface Definition (Informative)

2872

Comment [DSR22]: Needs updating (issue #38 and new parameter types)

2873
2874

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

2875
2876
2877
2878
2879

The interface definition is provided as five operations where each operation takes three or more of the following input parameters. The input parameters are described here in terms of the normative text in section 5. In all of these parameters, the value empty string ("") is interpreted the same as the value null.

Parameter name	Description
QXRI	Query XRI as defined in section 6.1.1.
trustType	The value of the <code>trust</code> media type subparameter of the Resolution Output Format parameter as specified in Table 6 of section 3.3, whose behavior is defined in section 6.1.2.
followRefs	The value of the <code>refs</code> media type subparameter of the Resolution Output Format parameter as specified in Table 6 of section 3.3, whose behavior is defined in section 6.1.2.
sepType	Service Type as defined in section 6.1.3.
sepMediaType	Service Media Type as defined in section 6.1.4.

Comment [DSR23]: Needs updating

2880

2881
2882

The five operations correspond to the following combinations of values of the Resolution Output Format parameter and its `sep` (service endpoint) subparameter (section 6.1.2) as shown below.

2883

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

2884 Following is the API and descriptions of the five operations.

2885 1. Resolve Authority to XRDS

```
2886 int resolveAuthToXRDS(  
2887     in string QXRI, in string trustType, in boolean followRefs,  
2888     out string XRDS, out string errorContext);
```

- 2889 • Performs authority resolution only (sections 5 and 6) and outputs the XRDS as specified in
2890 section 4.2.1 when the `sep` subparameter is `false`.
- 2891 • Only the authority segment of the QXRI is processed by this function. If the QXRI contains a
2892 path or query component, it is ignored.
- 2893 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 2894 • Returns the error code. If error, then the `errorContext` output argument may contain additional
2895 error information. The output XRDS will contain a final XRD with the same status code and
2896 optional context information in its `xrd:Status` element.

2897

2898 2. Resolve Authority to XRD

```
2899 int resolveAuthToXRD(  
2900     in string QXRI, in string trustType, in boolean followRefs,  
2901     out string XRD, out string errorContext);
```

- 2902 • Performs authority resolution only (sections 5 and 6) and outputs the final XRD as specified
2903 in section 4.2.2 when the `sep` subparameter is `false`.
- 2904 • Only the authority segment of the QXRI is processed by this function. If the QXRI contains a
2905 path or query component, it is ignored.
- 2906 • The output XRD argument will be signed or not depending on the value of `trustType`.
- 2907 • Returns the error code. If error, then the `errorContext` output argument may contain
2908 additional error information. The output XRD will contain the same status code and optional
2909 context information in its `xrd:Status` element.

2910

2911 3. Resolve Service Endpoint to XRDS

```
2912 int resolveSEPToXRDS(  
2913     in string QXRI, in string trustType, in string sepType,  
2914     in string sepMediaType, in boolean followRefs,  
2915     out string XRDS, out string errorContext);
```

- 2916 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
2917 and outputs the XRDS as specified in section 4.2.1 when the `sep` subparameter is `true`.
- 2918 • As specified in section 4.2.1, the output XRDS document will contain a nested XRDS
2919 document as the final child element if reference processing was necessary to locate the
2920 request service endpoint and the `followRefs` flag was set to `true`.
- 2921 • The final XRD in the output XRD will either contain at least one instance of the requested
2922 service endpoint or an error. IMPORTANT: Although the resolver will perform this verification,
2923 the final XRD is NOT filtered when the Resolution Output Format is an XRDS document.
2924 Filtering is only performed when the Resolution Output Format is an XRD document (see
2925 next section).
- 2926 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 2927 • Returns the error code. If error, then the `errorContext` output argument may contain
2928 additional error information. The output XRDS will contain a final XRD with the same status
2929 code and optional context information in its `xrd:Status` element.
- 2930 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
2931 same as the value `null`.

2932

2933 4. Resolve Service Endpoint to XRD

```
2934 int resolveSEPToXRD(  
2935     in string QXRI, in string trustType, in string sepType,  
2936     in string sepMediaType, in boolean followRefs,  
2937     out string XRDS, out string errorContext);
```

- 2938 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
2939 and outputs the XRD as specified in section 4.2.2 when the `sep` subparameter is `true`.
- 2940 • As specified in section 4.2.2, all elements in the output XRD subject to the global
2941 `xrd:priority` attribute will be returned in order of highest to lowest.
- 2942 • The output XRD will either contain at least one instance of the requested service endpoint or
2943 an error.
- 2944 • The output XRD will be *not* be signed regardless of the value of `trustType` because the
2945 XRD will be filtered to only the selected service endpoints.
- 2946 • Returns the error code. If error, then the `errorContext` output argument may contain
2947 additional error information. The output XRD will contain the same status code and optional
2948 context information in its `xrd:Status` element.
- 2949 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
2950 same as the value `null`.

2951 **5. Resolve Service Endpoint to URI List**

```
2952 int resolveSepToURIList(  
2953     in string QXRI, in string trustType, in string sepType,  
2954     in string sepMediaType, in boolean followRefs,  
2955     out string[] URIList, out string errorContext);
```

- 2956 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
2957 and, upon success, outputs a non-empty URI List as specified in section 4.2.3.
- 2958 • Returns the error code. If error, then the output URI List will be empty, and the
2959 `errorContext` output argument may contain additional error information.
- 2960 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
2961 same as the value null.