



Extensible Resource Identifier (XRI) Resolution Version 2.0

Working Draft 11, ED 04

5 September 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.....	9
1.1	Overview of XRI Resolution Architecture.....	9
1.2	Structure of this Specification.....	11
1.3	Examples of XRI Resolution Requests and Responses.....	12
1.4	Terminology and Notation.....	12
1.5	Normative References.....	13
1.6	Non-Normative References.....	14
2	Conformance.....	15
2.1	Conformance Targets.....	15
2.2	XRDS Clients.....	15
2.3	XRDS Servers.....	15
2.4	XRI Resolvers.....	15
2.4.1	Local Resolvers.....	15
2.4.2	Proxy Resolvers.....	15
2.5	XRI Authority Servers.....	15
3	Namespaces.....	16
3.1	XRI Namespaces for XRI Resolution.....	16
3.1.1	XRIs Reserved for XRI Resolution.....	16
3.1.2	XRIs Assigned to XRI Resolution Service Types.....	16
3.2	XML Namespaces for XRI Resolution.....	16
3.3	Media Types for XRI Resolution.....	17
4	XRDS Documents.....	19
4.1	XRDS and XRD Namespaces.....	19
4.2	XRD Elements and Attributes.....	19
4.2.1	Management Elements.....	21
4.2.2	Authority Trust Elements.....	21
4.2.3	Authority Synonym Elements.....	22
4.2.4	Service Endpoint Descriptor Elements.....	22
4.2.5	Service Endpoint Synonym Elements.....	23
4.2.6	Service Endpoint Trust Elements.....	23
4.2.7	Service Endpoint Selection Elements.....	24
4.3	XRD Attribute Processing Rules.....	24
4.3.1	ID Attribute.....	24
4.3.2	Version Attribute.....	24
4.3.3	Priority Attribute.....	25
4.4	XRI and IRI Encoding Requirements.....	25
5	XRDS Synonyms.....	26
5.1	Background.....	26
5.1.1	Hierarchical and Polyarchical Graph Models.....	26
5.1.2	Local and Global Synonyms.....	27
5.1.3	Canonical Synonyms.....	28
5.1.4	Synonym Verification.....	29

5.2 XRDS Synonym Elements	29
5.2.1 LocalID.....	29
5.2.2 CanonicalID	30
5.2.3 EquivID	30
5.2.4 CanonicalEquivID	30
5.2.5 Ref	31
5.3 Synonym Selection Rules	31
6 Discovering an XRDS Document from an HTTP(S) URI	32
6.1 Overview	32
6.2 HEAD Protocol.....	32
6.3 GET Protocol.....	32
7 XRI Resolution Inputs and Outputs.....	34
7.1 Inputs	34
7.1.1 QXRI (Authority String, Path String, and Query String)	35
7.1.2 Resolution Output Format	35
7.1.3 Service Type	36
7.1.4 Service Media Type	36
7.2 Outputs	37
7.2.1 XRDS Document.....	37
7.2.2 XRD Document.....	37
7.2.3 URI List.....	38
7.2.4 HTTP(S) Redirect	38
8 Generic Authority Resolution	39
8.1 XRI Authority Resolution	39
8.1.1 Service Type and Service Media Type.....	39
8.1.2 Protocol.....	40
8.1.3 Community Root Authorities	42
8.1.4 Self-Describing XRDS Documents.....	42
8.1.5 Qualified Subsegments.....	43
8.1.6 Cross-References	44
8.1.7 Recursing Authority Resolution.....	44
8.1.8 Construction of the Next Authority URI	45
8.2 IRI Authority Resolution.....	45
8.2.1 Service Type and Media Type	46
8.2.2 Protocol.....	46
8.2.3 Optional Use of HTTPS.....	46
9 Trusted Authority Resolution.....	47
9.1 HTTPS	47
9.1.1 Service Type and Service Media Type.....	47
9.1.2 Protocol.....	47
9.2 SAML	47
9.2.1 Service Type and Service Media Type.....	48
9.2.2 Protocol.....	48
9.2.3 Recursing Authority Resolution.....	49
9.2.4 Client Validation of XRDS.....	49

9.2.5 Correlation of ProviderID and KeyInfo Elements	50
9.3 HTTPS+SAML	51
9.3.1 Service Type and Service Media Type	51
9.3.2 Protocol	51
10 Proxy Resolution	52
10.1 Service Type and Media Types	52
10.2 HXRI	52
10.3 HXRI Query Parameters	54
10.4 HTTP(S) Accept Headers	55
10.5 Null Resolution Output Format	55
10.6 Outputs and HTTP(S) Redirects	55
10.7 Differences Between Proxy Resolution Servers	56
10.8 Combining Authority and Proxy Resolution Servers	56
11 Service Endpoint Selection	57
11.1 Processing Rules	57
11.2 Service Endpoint Selection Logic	59
11.3 Selection Element Matching Rules	60
11.3.1 Selection Element Match Options	61
11.3.2 The Match Attribute	61
11.3.3 Absent Selection Element Matching Rule	61
11.3.4 Empty Selection Element Matching Rule	62
11.3.5 Multiple Selection Element Matching Rule	62
11.3.6 Type Element Matching Rules	62
11.3.7 Path Element Matching Rules	62
11.3.8 MediaType Element Matching Rules	63
11.4 Service Endpoint Matching Rules	63
11.4.1 Service Endpoint Match Options	63
11.4.2 Select Attribute Match Rule	63
11.4.3 All Positive Match Rule	63
11.4.4 Default Match Rule	64
11.5 Service Endpoint Selection Rules	64
11.5.1 Positive Match Rule	64
11.5.2 Default Match Rule	64
11.6 Pseudocode	64
11.7 Construction of Service Endpoint URIs	66
12 Reference Processing	68
12.1 Authority References	68
12.1.1 Processing Rules	68
12.1.2 Nesting XRDS Documents	70
12.2 Service References	71
12.2.1 Processing Rules	71
12.2.2 Adding XRD Documents	73
13 Synonym Verification	75
13.1 CanonicalID Verification	75
13.1.1 HTTP(S) URI Verification Rules	76

13.1.2 XRI Verification Rules	76
13.2 CanonicalEquivID Verification	76
13.3 Verification Error Codes and Recommended Identifiers	77
13.4 Examples	78
13.4.1 CanonicalID Verification	78
13.4.2 CanonicalEquivID Verification	79
13.5 EquivID Verification	81
14 Error Processing	82
14.1 Error Codes	82
14.2 Error Context Strings	84
14.3 Error Handling in Recursing and Proxy Resolution	84
15 Use of HTTP(S)	86
15.1 HTTP Errors	86
15.2 HTTP Headers	86
15.2.1 Caching	86
15.2.2 Location	86
15.2.3 Content-Type	86
15.3 Other HTTP Features	86
15.4 Caching and Efficiency	87
15.4.1 Resolver Caching	87
15.4.2 Synonyms	87
16 Extensibility and Versioning	88
16.1 Extensibility	88
16.1.1 Extensibility of XRDs	88
16.1.2 Other Points of Extensibility	88
16.2 Versioning	89
16.2.1 Version Numbering	89
16.2.2 Versioning of the XRI Resolution Specification	89
16.2.3 Versioning of XRDs	89
16.2.4 Versioning of Protocols	90
17 Security and Data Protection	91
17.1 DNS Spoofing or Poisoning	91
17.2 HTTP Security	91
17.3 SAML Considerations	91
17.4 Limitations of Trusted Resolution	91
17.5 Community Root Authorities	92
17.6 Caching Authorities	92
17.7 Recursing and Proxy Resolution	92
17.8 Denial-Of-Service Attacks	92
A. Acknowledgments	93
B. Non-Normative Text	94
C. Revision History	95
D. XML Schema for XRDS and XRD (Normative)	96
E. RelaxNG Compact Syntax Schema for XRDS and XRD (Informative)	99
F. Media Type Definition for application/xrds+xml (Normative)	100

G. Media Type Definition for application/xrd+xml (Normative) 101
H. Example Local Resolver Interface Definition (Informative)..... 102

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	resolver	resolver
Resolution server	authoritative nameserver	authority resolution service
Recurring resolution	recurring nameserver	recurring authority resolution service or proxy resolver

Deleted: local

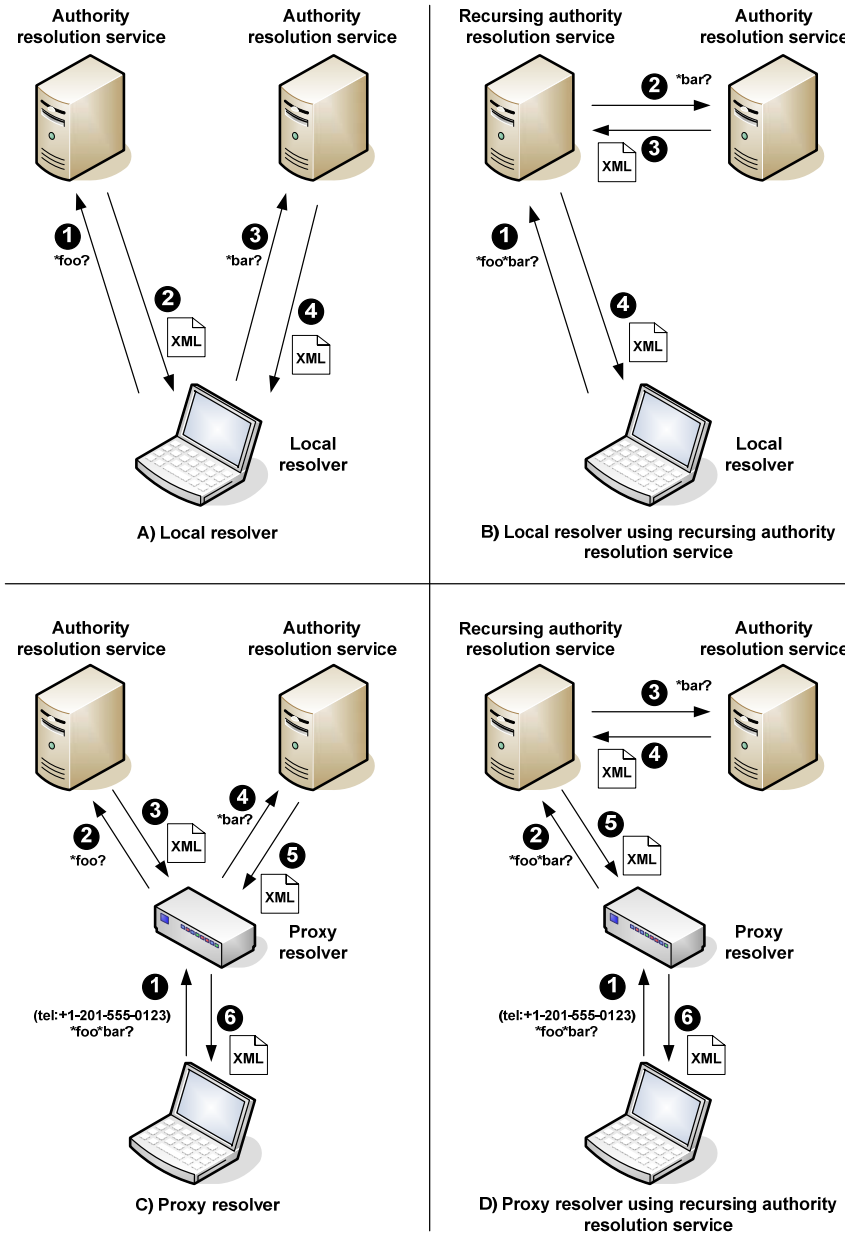
Deleted: local

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 an HTTP(S) interface for a local XRI
 31 resolver (one implemented using a platform-specific API). Proxy resolvers enable applications—
 32 even those that do not natively understand XRIs but can process HTTP URIs—to easily access
 33 the functions of an XRI resolver remotely.

Deleted: with an HTTP(S) interface

34 Figure 1 shows four scenarios of how these components might interact to resolve
 35 `xri://(tel:+1-201-555-0123)*foo*bar` (note that, unlike DNS, this works from left-to-
 36 right).



37

38 *Figure 1: Four typical scenarios for XRI authority resolution.*

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

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

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

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

Deleted: servers

81 1.2 Structure of this Specification

82 This specification is structured into the following major sections:

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

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

118 1.3 Examples of XRI Resolution Requests and Responses

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

122 1.4 Terminology and Notation

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

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

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

131 Formatting conventions used in this document:

132 Examples look like this.

133 ABNF productions look like this.

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

135 1.5 Normative References

- 136 **[DNSSEC]** D. Eastlake, *Domain Name System Security Extensions*,
137 <http://www.ietf.org/rfc/rfc2535>, IETF RFC 2535, March 1999.
- 138 **[RFC2045]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
139 *Part One: Format of Internet Message Bodies*,
140 <http://www.ietf.org/rfc/rfc2045.txt>, IETF RFC 2045, November 1996.
- 141 **[RFC2046]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
142 *Part Two: Media Types*, <http://www.ietf.org/rfc/rfc2046.txt>, IETF RFC
143 2046, November 1996.
- 144 **[RFC2119]** S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*,
145 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 146 **[RFC2141]** R. Moats, *URN Syntax*, <http://www.ietf.org/rfc/rfc2141.txt>, IETF RFC
147 2141, May 1997.
- 148 **[RFC2483]** M. Mealling, R. Daniel Jr., *URI Resolution Services Necessary for URN*
149 *Resolution*, <http://www.ietf.org/rfc/rfc2483.txt>, IETF RFC 2483, January
150 1999.
- 151 **[RFC2616]** R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.
152 Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*,
153 <http://www.ietf.org/rfc/rfc2616.txt>, IETF RFC 2616, June 1999.
- 154 **[RFC2818]** E. Rescorla, *HTTP over TLS*, <http://www.ietf.org/rfc/rfc2818.txt>, IETF
155 RFC 2818, May 2000.
- 156 **[RFC3023]** M. Murata, S. St-Laurent, D. Kohn, *XML Media Types*,
157 <http://www.ietf.org/rfc/rfc3023.txt>, IETF RFC 3023, January 2001.
- 158 **[RFC3986]** T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier*
159 *(URI): Generic Syntax*, <http://www.ietf.org/rfc/rfc3986.txt>, IETF RFC
160 3986, January 2005.
- 161 **[RFC4234]** D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications:*
162 *ABNF*, <http://www.ietf.org/rfc/rfc4234.txt>, IETF RFC 4234, October 2005.
- 163 **[RFC4288]** N. Freed, J. Klensin, *Media Type Specifications and Registration*
164 *Procedures*, <http://www.ietf.org/rfc/rfc4288.txt>, IETF RFC 4288,
165 December 2005.
- 166 **[SAML]** S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for*
167 *the OASIS Security Assertion Markup Language (SAML) V2.0*,
168 <http://www.oasis-open.org/committees/security>, March 2005.
- 169 **[Unicode]** The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined
170 by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley,
171 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1
172 (<http://www.unicode.org/versions/Unicode4.0.1>) and by Unicode 4.1.0
173 (<http://www.unicode.org/versions/Unicode4.1.0>), March, 2005.
- 174 **[UUID]** Open Systems Interconnection – *Remote Procedure Call*, ISO/IEC
175 11578:1996, <http://www.iso.org/>, August 2001.
- 176 **[XML]** T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau,
177 *Extensible Markup Language (XML) 1.0, Third Edition*, World Wide Web
178 Consortium, <http://www.w3.org/TR/REC-xml/>, February 2004.

179	[XMLDSig]	D. Eastlake, J. Reagle, D. Solo et al., <i>XML-Signature Syntax and Processing</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlsig-core/ , February, 2002.
180		
181		
182	[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.
183		
184		
185	[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.
186		
187		
188	[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.
189		
190		
191	[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.
192		
193		
194	[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.
195		
196		

197 1.6 Non-Normative References

198	[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.
199		
200	[XRIGuide]	[TODO]
201	[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.
202		
203		
204		
205		
206	[Yadis]	TODD

207 **2 Conformance**

208 This section specifies conformance and conformance claims.

209 **2.1 Conformance Targets**

210 The targets of this specification are:

- 211 1. XRDS clients, which provided a limited subset of the functionality of XRI resolvers.
- 212 2. XRDS servers, which provided a limited subset of the functionality of XRI authority
213 servers.
- 214 3. XRI resolvers, which may be either local resolvers or proxy resolvers.
- 215 4. XRI authority servers.

216 Note that a single implementation may serve all functions, i.e., an XRI authority server may also
217 function as an XRDS client and server and an XRI local and proxy resolver.

218 **2.2 XRDS Clients**

219 [TODO]

220 **2.3 XRDS Servers**

221 [TODO]

222 **2.4 XRI Resolvers**

223 **2.4.1 Local Resolvers**

224 [TODO]

225 **2.4.2 Proxy Resolvers**

226 [TODO]

227 **2.5 XRI Authority Servers**

228 [TODO]

229

3 Namespaces

230

3.1 XRI Namespaces for XRI Resolution

231

232

233

234

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

235

3.1.1 XRIs Reserved for XRI Resolution

236

237

The XRIs in Table 2 are assigned by this specification for the purposes of XRI resolution and 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

238

Table 2: XRIs reserved for XRI resolution.

239

3.1.2 XRIs Assigned to XRI Resolution Service Types

240

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	8
xri://\$res*auth*(\$v*2.0)	Version 2.0 of above	8
xri://\$res*proxy	HTTP(S) proxy resolution service	10
xri://\$res*proxy*(\$v*2.0)	Version 2.0 of above	10

241

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

242

243

244

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

245

3.2 XML Namespaces for XRI Resolution

246

247

Throughout this document, the following XML namespace prefixes have the meanings defined in 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

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

249 3.3 Media Types for XRI Resolution

250 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as
 251 defined by [RFC2046], particularly in HTTP Accept headers as specified in [RFC2616]. Table 5
 252 specifies the media types used for XRI resolution. Note that in XRI authority resolution, these
 253 media types MUST be passed as HTTP Accept header values. By contrast, in XRI proxy resolution
 254 these media types MUST be passed as query parameters in an HTTP(S) URI as specified in
 255 section 10.

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 11	Section 5 of [RFC2483]

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

257 To provide full control of XRI resolution, the media types specified in Table 5 accept the media
 258 type parameters defined in Table 6. Note that when these media type parameters are appended
 259 to a media type in the XRI proxy resolver interface, the semicolon character used to concatenate
 260 them must be percent-encoded as specified in section [TODO].

Comment [DSR9]: Normative text requiring percent-encoding of semicolon in HXRI query parameters must still be added to section 8.2

Parameter	Values	Usage	See Section
https	true false	Specifies use of HTTPS trusted resolution	9.1
saml	true false	Specifies use of SAML trusted resolution	9.2
sep	true false	Specifies whether service endpoint selection should be performed	11
nodefaut_t	true false	Specifies whether a default match on a Type service endpoint selection element is allowed	11.3

nodefault_p	true false	Specifies whether a default match on a Path service endpoint selection element is allowed	11.3
nodefault_m	true false	Specifies whether a default match on a MediaType service endpoint selection element is allowed	11.3
refs	true false	Specifies whether references should be followed during resolution	12
cid	true false	Specifies whether canonical ID verification must be performed	13

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

262 See sections 7 - 13 for more about usage of these media types and parameters.

263

4 XRDS Documents

264
265
266
267
268

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.

269

4.1 XRDS and XRD Namespaces

270
271
272
273
274

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.

275
276
277
278
279

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].

280
281
282
283

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.

284

4.2 XRD Elements and Attributes

285
286

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

287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312

```
<XRDS xmlns="xri://$xrds" ref="xri://(http%3A%2F%2Fexample.org)*foo">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*foo</Query>
    <Status code="100"/>
    <Expires>2005-05-30T09:30:10Z</Expires>
    <ProviderID>
      xri://(http%3A%2F%2Fexample.org)!1234
    </ProviderID>
    <LocalID>*baz</LocalID>
    <CanonicalID>
      xri://(https%3A%2F%2Fexample.org)!1234!5678
    </CanonicalID>
    <EquivID>
      xri://(http%3A%2F%2Fexample.org)!1234!5678
    </EquivID>
    <CanonicalEquivID>
      xri://=!4a76!c2f7!9033.78bd
    </CanonicalEquivID>
    <Ref>xri://@!4a76!c2f7!9033.78bd</Ref>
    <Service>
      <ProviderID>
        xri://(https%3A%2F%2Fexample.org)!1234
      </ProviderID>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <MediaType>application/xrds+xml</MediaType>
      <URI priority="10">http://resolve.example.com</URI>
```

Deleted:
<ProviderID>xri://!1000!1234.5678</ProviderID>¶

```

313 <URI priority="15">http://resolve2.example.com</URI>
314 <URI>https://resolve.example.com</URI>
315 </Service>
316 <Service>
317 <ProviderID>
318 <xri://(https%3A%2F%2Fexample.org)!1234
319 </ProviderID>
320 <Type>xri://$res*auth*($v*2.0)</Type>
321 <MediaType>application/xrds+xml;https=true</MediaType>
322 <URI>https://resolve.example.com</URI>
323 </Service>
324 <Service>
325 <Type match="null" />
326 <Path select="true">media/pictures</Path>
327 <MediaType select="true">image/jpeg</MediaType>
328 <URI append="path" >http://pictures.example.com</URI>
329 </Service>
330 <Service>
331 <Type match="null" />
332 <Path select="true">media/videos</Path>
333 <MediaType select="true">video/mpeg</MediaType>
334 <URI append="path" >http://videos.example.com</URI>
335 </Service>
336 <Service>
337 <ProviderID> xri://!!1000!1234.5678</ProviderID>
338 <Type match="null" />
339 <Path match="default" />
340 <URI>http://example.com/local</URI>
341 </Service>
342 <Service>
343 <Type>http://example.com/some/service/v3.1</Type>
344 <URI>http://example.com/some/service/endpoint</URI>
345 </Service>
346 </XRD>
347 </XRDS>

```

Deleted:
 <ProviderID>xri://!!1000!123
 4.5678</ProviderID>¶

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

351 4.2.1 Management Elements

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

354 **xrd:XRD**

355 Container element for all other XRD elements. Includes an OPTIONAL `xml:id` attribute
356 of type `xs:ID`. This attribute is REQUIRED in trusted resolution to uniquely identify this
357 element within the containing `xrd:XRDS` document. It also includes an OPTIONAL
358 `xrd:idref` attribute of type `xs:idref`. This attribute is REQUIRED in trusted resolution
359 when an XRD element in a nested `xrd:XRDS` document must reference a previously
360 included XRD instance. See sections 4.3 and 12.1. Lastly, it includes an `xrd:version`
361 attribute that is OPTIONAL for uses outside of XRI resolution but REQUIRED for XRI
362 resolution as defined in section 4.3.2

363 **xrd:XRD/xrd:Query**

364 0 or 1 per `xrd:XRD` element. Expresses the XRI, IRI, or URI reference in URI-normal
365 form whose resolution results in this `xrd:XRD` element. For XRI authority resolution, this
366 must be a qualified subsegment of the authority component of the query XRI.

367 **xrd:XRD/xrd:Status**

368 0 or 1 per `xrd:XRD` element. Contains a REQUIRED attribute `xrd:code` of type
369 `xs:int` that provides a numeric status code. The contents of the element are a human-
370 readable message string describing the status of the response. For XRI resolution,
371 values of the Status element and `xrd:code` attribute are defined in section 14.

372 **xrd:XRD/xrd:Expires**

373 0 or 1 per `xrd:XRD` element. The date/time, in the form of `xs:dateTime`, after which
374 this XRD cannot be relied upon. To promote interoperability, this date/time value
375 SHOULD use the UTC "Z" time zone and SHOULD NOT use fractional seconds. A
376 resolver using this XRD MUST NOT use the XRD after the time stated here. A resolver
377 MAY discard this XRD before the time indicated in this result. If the HTTP transport
378 caching semantics specify an expiry time earlier than the time expressed in this attribute,
379 then a resolver MUST NOT use this XRD after the expiry time declared in the HTTP
380 headers per section 13.2 of [RFC2616]. See section 15.2.1.

381 4.2.2 Authority Trust Elements

382 The second set of elements are for applications where trust must be established in the authority
383 providing the XRD. These elements are OPTIONAL for generic authority resolution (section 8),
384 but REQUIRED for specific types of trusted authority resolution (section 9).

385 **xrd:XRD/xrd:ProviderID**

386 0 or 1 per `xrd:XRD`. A unique identifier of type `xs:anyURI` for the authority producing
387 this XRD. The value of this element MUST be a persistent identifier. There MUST be
388 negligible probability that the value of this element will be assigned as an identifier to any
389 other authority. For purposes of CanonicalID verification (section 12), it is
390 RECOMMENDED to use a fully persistent XRI as defined in [XRISyntax]. If a URN
391 [RFC2141] or other persistent identifier is used, it is RECOMMENDED to express it as an
392 XRI cross-reference as defined in [XRISyntax]. Note that for XRI authority resolution, the
393 authority identified by this element is the *parent* authority (the provider of the current
394 XRD), not the *child* authority (the target of the current XRD). The latter is identified by the
395 `xrd:XRD/xrd:Service/xrd:ProviderID` element inside a resolution service
396 endpoint (see below).

397 **xrd:XRD/saml:Assertion**

398 0 or 1 per `xrd:XRD`. A SAML assertion from the *parent* authority (the provider of the
399 current XRD) that asserts that the information contained in the current XRD is
400 authoritative. Because the assertion is digitally signed and the digital signature
401 encompasses the containing `xrd:XRD` element, it also provides a mechanism for the
402 recipient to detect unauthorized changes since the time the XRD was published.

403 Note that while a `saml:Issuer` element is required within a `saml:Assertion` element,
404 this specification makes no requirement as to the value of the `saml:Issuer` element. It
405 is up to the XRI community resolution root to place restrictions, if any, on the
406 `saml:Issuer` element. A suitable approach is to use an XRI in URI-normal form that
407 identifies the community root authority. See section 8.1.3.

Comment [DSR10]: This section was rewritten again in ED04 and needs complete review.

408 4.2.3 Authority Synonym Elements

409 In the context of XRI architecture, two identifiers are *synonyms* if they are not character-for-
410 character equivalent but identify the same target resource ([the resource to which the identifier](#)
411 [was assigned by the identifier authority](#)). XRDS synonym architecture and the normative rules for
412 synonym usage in XRI resolution are specified in section 5. All synonym elements with zero-or-
413 more cardinality elements accept the optional global `xrd:priority` attribute (see section 4.3.3).

414 `xrd:XRD/xrd:LocalID`

415 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be assigned by the same
416 parent authority providing the current XRD. Asserts a interchangeable synonym for the
417 value of the `xrd:Query` element. See section 5.2.1.

418 `xrd:XRD/xrd:CanonicalID`

419 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier. Asserts
420 the hierarchical canonical synonym for the QXRI. See section 5.2.2. MAY be verified as
421 defined in section 13.

422 `xrd:XRD/xrd:EquiVID`

423 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier.
424 Asserts either: a hierarchical global synonym for the QXRI (besides the CanonicalID), or
425 b) a polyarchical global synonym for the QXRI. See section 5.2.3. MAY be used for
426 CanonicalEquiVID verification as defined in section 13.

427 `xrd:XRD/xrd:CanonicalEquiVID`

428 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier. Asserts
429 the polyarchical canonical synonym for the QXRI. See section 5.2.4. MAY be verified as
430 defined in section 13.

431 `xrd:XRD/xrd:Ref`

432 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier.
433 MAY be processed to locate additional XRDS documents describing the child authority as
434 defined in section 12. See section 5.2.5.

Deleted: Management

435 4.2.4 Service Endpoint Descriptor Elements

436 The next set of elements are used to describe service endpoints—the set of network endpoints
437 advertised in an XRD for performing further resolution, obtaining further metadata, or interacting
438 directly with the described resource. Again, because there can be more than one instance of a
439 service endpoint that satisfies a service endpoint selection query, or more than one instance of a
440 service endpoint URI, these elements both have the global `xrd:priority` attribute (see section
441 4.3.3).

442 `xrd:XRD/xrd:Service`

443 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.
444 Referred to by the abbreviation *SEP*.

445 **`xrd:XRD/xrd:Service/xrd:URI`**

446 0 or more per `xrd:XRD/xrd:Service` element. Type `xs:anyURI`. If present, it
447 indicates a transport-level URI for access to the capability described by the parent
448 Service element. For the service types defined for XRI resolution in section 3.1.2, this
449 URI MUST be an HTTP or HTTPS URI. Other services may use other transport
450 protocols. This element includes two attributes, `xrd:append` and `xrd:uric`, that
451 govern construction of the final service endpoint URI. See section 11.7.

Deleted: an

Deleted: attribute

Deleted: s

452 **4.2.5 Service Endpoint Synonym Elements**

453 These are similar to the authority synonym elements except they provide synonyms used in the
454 context of a specific service endpoint. The normative rules for use of synonyms in XRI resolution
455 are specified in section 5. These elements all have the optional global `xrd:priority` attribute
456 (see section 4.3.3).

Deleted: for the child authority

457 **`xrd:XRD/xrd:Service/xrd:LocalID`**

458 0 or more per `xrd:XRD/xrd:Service` element. Identical to `xrd:XRD/xrd:LocalID`
459 above except the synonym is assigned by the provider of the service and not the
460 authority for the XRD. MAY be used to provide one or more identifiers by which the target
461 resource SHOULD be identified in the context of the containing service endpoint.

462 **`xrd:XRD/xrd:Service/xrd:Ref`**

463 0 or more per `xrd:XRD/xrd:Service` element. MUST be an absolute identifier.
464 Identical to `xrd:XRD/xrd:Ref` except processed only in the context of service
465 selection. MAY be processed to locate additional XRDS documents with additional
466 service endpoint metadata describing the target resource as defined in section 12.2.

467 **4.2.6 Service Endpoint Trust Elements**

468 Similar to the authority trust elements, these elements enable trust to be established in the
469 provider of the service endpoint. These elements are OPTIONAL for generic authority resolution
470 (section 8), but REQUIRED for SAML trusted authority resolution (section 9.2).

471 **`xrd:XRD/xrd:Service/xrd:ProviderID`**

472 0 or 1 per `xrd:XRD/xrd:Service` element. Identical to the
473 `xrd:XRD/xrd:ProviderID` above, except this identifies the provider of the *described*
474 *service endpoint* instead of the provider of the current XRD. In SAML trusted resolution,
475 when a resolution request is made to the authority at this service endpoint, the contents
476 of the `xrd:XRD/xrd:ProviderID` element in the response MUST match the content of
477 this element for correlation. See section 9.2.5. The same usage MAY apply to other
478 services not defined in this specification, however that is beyond the scope of this
479 specification. Authors of other specifications employing XRD service endpoints SHOULD
480 define the scope and usage of this element, particularly for trust verification.

481 **`xrd:XRD/xrd:Service/ds:KeyInfo`**

482 0 or 1 per `xrd:XRD/xrd:Service` element. Provides the digital signature metadata
483 necessary to validate an XRD provided as a resolution response by the described
484 authority. This element comprises the key distribution method for SAML trusted authority
485 resolution in the XRI resolution framework—see section 9.2.5. The same usage MAY
486 apply to other services not defined in this specification.

487 4.2.7 Service Endpoint Selection Elements

488 The final set of elements are used in XRI resolution to select service endpoints. They include two
489 global attributes used for this purpose: `xrd:match` and `xrd:select`. See sections 11.2 and
490 11.4.

491 `xrd:XRD/xrd:Service/xrd:Type`

492 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI`
493 that identifies the type of capability available at this service endpoint. See section 3.1.2
494 for the resolution service types defined in this specification. If a service endpoint does not
495 include at least one `xrd:Type` element, the service type is effectively described by the
496 type of URI specified in the `xrd:XRD/xrd:Service/xrd:URI` element, i.e., an HTTP
497 URI specifies an HTTP service.

498 `xrd:XRD/xrd:Service/xrd:MediaType`

499 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of
500 content available at this service endpoint. The value of this element MUST be of the form
501 of a media type defined in [RFC2046]. See section 3.3 for the media types used in XRI
502 resolution.

503 `xrd:XRD/xrd:Service/xrd:Path`

504 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string
505 value meeting the `xri-path` production defined in section 2.2.3 of [XRISyntax].

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

510 4.3 XRD Attribute Processing Rules

511 4.3.1 ID Attribute

512 For uses such as XRI trusted resolution (section 9.2) that require unique identification of multiple
513 XRD elements within an XRDS document, the XRD element uses an optional `xml:id` attribute
514 as defined by the W3C XML ID specification [XMLID]. If present, the value of this element MUST
515 be unique for all elements in the containing XML document. Because an XRI resolver may need
516 to assemble multiple XRDs received from different authority resolution services into one XRDS
517 document, there MUST be negligible probability that the value of the `xrd:XRD/@xml:id`
518 attribute is not globally unique. For this reason the value of this attribute SHOULD be a UUID as
519 defined by [UUID] prefixed by a single underscore character "_" in order to make it a legal
520 `NCName` as required by [XMLID]. However the value of this attribute MAY be generated by any
521 algorithm that fulfills the same requirements of global uniqueness and `NCName` conformance.

522 Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their
523 XML document order MUST match the order in which they were resolved (see section 8.1.2).

524 Also, if reference processing requires the same XRD to be included in an XRDS document twice
525 (via a nested XRDS document), that XRD MUST reference the previous instance using the
526 `xrd:XRD/@xml:idref` attribute as defined in section 12.1.2.

527 4.3.2 Version Attribute

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

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

533 4.3.3 Priority Attribute

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

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

- 542 5. The client SHOULD select the element instance with the lowest numeric value of the
543 priority attribute. For example, an element with priority attribute value of “10” should be
544 selected before an element with a priority attribute value of “11”, and an element with
545 priority attribute value of “11” should be selected before an element with a priority
546 attribute value of “25”. Zero is the highest priority attribute value. Null is the lowest priority
547 attribute value—it is the equivalent of a value of infinity. It is RECOMMENDED to use a
548 large finite value (100 or more) rather than a null value.
- 549 6. If an element has no priority attribute, its priority attribute value is considered to be null,
550 i.e., the lowest possible priority value. Rather than omitting a priority attribute, it is
551 RECOMMENDED that XRI authorities follow the standard practice in DNS and set the
552 default priority attribute value to “10”.
- 553 7. If two or more instances of the same element type have identical priority attribute values
554 (including the null value), the client SHOULD select one of the instances at random. This
555 client SHOULD NOT simply choose the first instance that appears in XML document
556 order (this is important in order to support intentional load balancing).
- 557 8. An element selected according to these rules is referred to in this specification as “the
558 highest priority element”. If this element is subsequently disqualified from the set of
559 qualified elements, the next element selected according to these rules is referred to as
560 “the next highest priority element”. If an XRI resolution operation specifying selection of
561 the highest priority element fails, the resolver SHOULD attempt to select the next highest
562 priority element unless otherwise specified. This process SHOULD be continued for all
563 other instances of the qualified elements until success is achieved or all instances are
564 exhausted.

565 4.4 XRI and IRI Encoding Requirements

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

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

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

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

Comment [DSR11]: This section has been completely rewritten and should be reviewed in its entirety.

579

5 XRDS Synonyms

580
581
582

XRDS architecture 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 7 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	Local	MUST NOT
CanonicalID	Zero-or-one	Hierarchical	Global	MUST NOT
EquivID	Zero-or-more	Hierarchical or polyarchical	Global	SHOULD
CanonicalEquivID	Zero-or-one	Hierarchical or polyarchical	Global	SHOULD
Ref	Zero-or-more	Hierarchical or polyarchical	Global	SHOULD

583

Table 7: The five XRD synonym elements.

584
585

This section describes the XRDS synonym model and specifies the rules for usage of each synonym element.

586

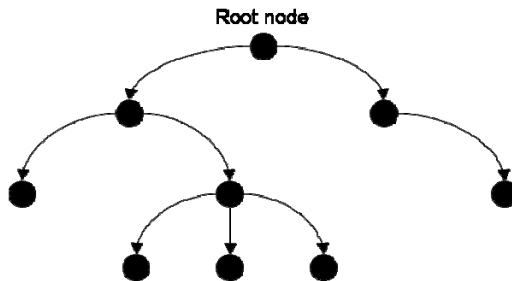
5.1 Background

587

5.1.1 Hierarchical and Polyarchical Graph Models

588
589
590

A pure hierarchical identifier model is a graph of parent-child relationships in which every child node is identified by exactly one arc from exactly one parent node. The set of arcs that connect any two nodes is called the *path* between the nodes.

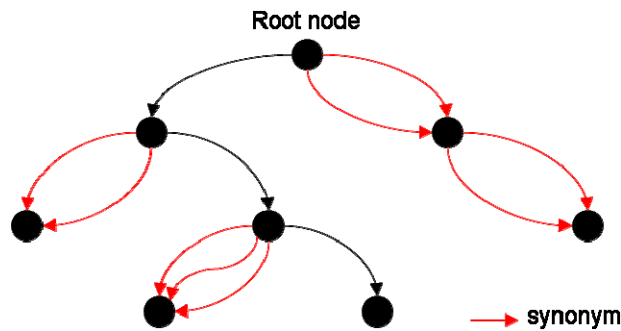


591
592

Figure 2: Hierarchical identifier graph.

593
594
595
596

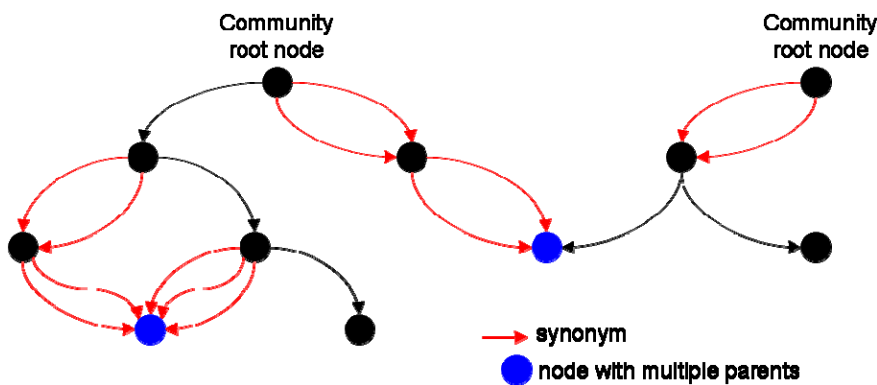
A hierarchical graph model can support synonyms by modifying the rules so that a child may have one-or-more arcs from exactly one parent. This is called a *hierarchical multigraph*. Each arc from the same parent to the same child is a synonym of the other arcs between that parent and that child as shown by the red arrows in Figure 3.



597

598 *Figure 3: Hierarchical multigraph with synonyms.*

599 A polyarchical multigraph takes the next step: a child node may have *one-or-more* arcs from *one-or-more* parent nodes. This means a child node may be a member of more than one hierarchy. It
 600 also means there may be multiple root nodes; in XRI architecture these are referred to as
 601 *community root nodes*. An example polyarchical multigraph is shown in Figure 4.
 602



603

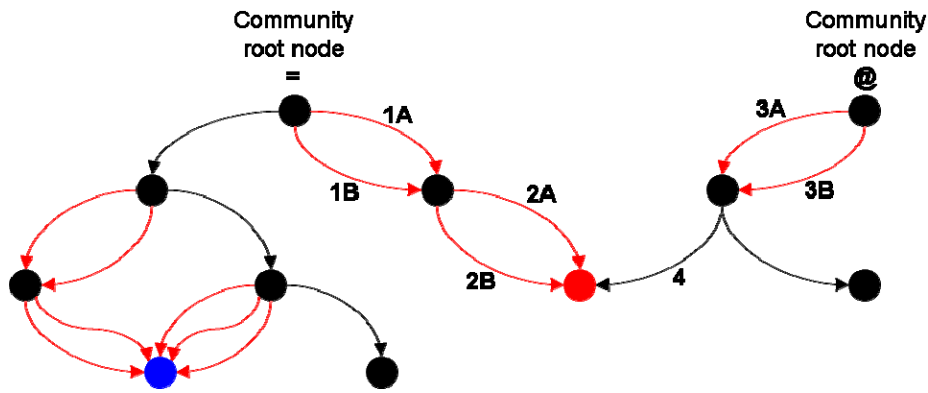
604 *Figure 4: Polyarchical multigraph*

605 The two nodes shown in blue have polyarchical identifiers because they can be identified via
 606 multiple paths from different parent nodes.

607 XRI architecture uses a polyarchical multigraph model because it reflects identification in the real
 608 world, where many resources have more than one identifier in more than one context. Because
 609 nodes in the XRI graph represent different identifier authorities (resources that assign identifiers
 610 to other resources), a parent node is called a *parent authority* and a child node is called a *child*
 611 *authority*. Note that in an XRD, the parent authority is self-identified by the
 612 `xrd:XRD/xrd:ProviderID` element (if present). See section 4.2.

613 5.1.2 Local and Global Synonyms

614 In both hierarchical and polyarchical multigraph models, the scope of a synonym may be either
 615 *local* or *global*. A local synonym is a relative identifier that identifies a single arc between a parent
 616 node and a child node. It is synonymous with another relative identifier if: a) both originate at the
 617 same parent node, and c) both terminate at the same child node. For example, in Figure 5, arcs
 618 **1A** and **1B** are a pair of local synonyms relative to the community root node =. Likewise arcs **2A**
 619 and **2B** are a pair of local synonyms relative to the parent node identified by arcs **1A** and **1B**, and
 620 arcs **3A** and **3B** are a pair of local synonyms relative to the community root node =.



621

622 *Figure 5: Examples of local and global synonyms.*

623 By contrast, a global synonym must be an absolute identifier, i.e., one that originates in a
 624 community root node. It is synonymous with another absolute identifier in the graph if both
 625 identifiers terminate at the same child node. Global synonyms fall into two categories:

- 626 1. *Hierarchical global synonyms* conform to the strict one-parent-per-child rules of a
 627 hierarchical graph model, i.e., they must both originate from the same root node and
 628 traverse the same set of child nodes before terminating in the same target child node. For
 629 this reason, hierarchical global synonyms always have the same number of arcs, and the
 630 product of the total number of local synonyms for each arc gives you the total number of
 631 hierarchical global synonyms that exist for any given path.
- 632 2. *Polyarchival global synonyms* conform to the looser rules of a polyarchival graph model,
 633 i.e., they may originate at any community root node and traverse any set of child nodes
 634 before terminating in the same target child node. Thus polyarchival synonyms are not
 635 required to have the same number of arcs.

636 In Figure 5, there are a total of six polyarchival global synonyms for the red target node. Four are
 637 hierarchical global synonyms originating at the community root node =: =1A*2A, =1A*2B, =1B*2A,
 638 and =1B*2B. Two more are hierarchical global synonyms originating at the community root node
 639 @: @3A*4 and @3B*4.

640 Polyarchival global synonyms are particularly useful for establishing and verifying relationships
 641 that span multiple trust domains.

642 5.1.3 Canonical Synonyms

643 In a pure hierarchical graph model, target nodes have a single global identifier. In hierarchical and
 644 polyarchival multigraph models, a target node may have any number of global synonyms. In this
 645 case, any parent node may assert that one of these global synonyms serves as the *canonical*
 646 *synonym* for a child node. Where possible, this is the synonym should be used by consuming
 647 applications to serve as the primary identifier of the child node.

648 As with global synonyms, a canonical synonym may be one of two types:

- 649 1. A *hierarchical canonical synonym* conforms to the strict rule that it must originate from the
 650 same community root node and traverse the same set of child nodes before terminating
 651 in the same target child node as the identifier to which it is synonymous.
- 652 2. A *polyarchival canonical synonym* conforms to the looser rule that it may originate at any
 653 community root node as long as it terminates at the same target node as the identifier to
 654 which it is synonymous.

655 These distinctions are important when it comes to certain trust models, as discussed in the next
 656 section.

657 5.1.4 Synonym Verification

658 When identifier systems are used to identify people—or any other resource that can serve as an
659 actor on a network—security policies typically require verification of the binding between the
660 identifier and the resource it represents. For example, a username must typically be
661 authenticated with a password before access is granted to an online account.

662 Thus if an identifier system supports synonyms and a synonym will be granted the same security
663 privileges as the query identifier, it is equally important to verify the binding between an identifier
664 and an asserted synonym.

665 Verification of hierarchical global synonyms is relatively straightforward, since it involves
666 verification of consecutive sets of local synonyms, each issued by one identifier authority in their
667 own local trust domain. Verification of polyarchical global synonyms is more complex because
668 each local synonym may be issued by a different identifier authority in a different trust domain.

669 XRI resolution architecture includes automated synonym verification for canonical synonyms. For
670 details see section 12, *Synonym Verification*.

671 5.2 XRDS Synonym Elements

672 In each XRD, the `xrd:Query` element is used to echo back the identifier for which a resolution
673 query was made. This is called the *query identifier*. The other XRD elements in this section assert
674 synonyms for the query identifier. Note that only one, `LocalID`, is a direct local synonym for the
675 query identifier itself. The others are all global synonyms for the complete query identifier that has
676 been resolved up to and including the current XRD.

677 5.2.1 LocalID

678 In XRD documents, a local synonym is asserted using the `xrd:LocalID` element. `LocalIDs` are
679 used at both the authority level of an XRD (as a child of the root `xrd:XRD` element) and the
680 service level (as a child of the root `xrd:XRD/xrd:Service` element).

681 At the authority level, the value of the `xrd:XRD/xrd:LocalID` element is a local synonym for
682 the query identifier. From a resolution standpoint, this means it is interchangeable with the
683 contents of the `xrd:Query` element. Therefore a `LocalID` MUST be assigned by the same
684 authority producing the current XRD and MUST identify the same child authority as the the
685 `xrd:Query` element. Resolution of a `LocalID` from the same parent authority MUST return the
686 same XRD as the XRD in which the `LocalID` appears (with the exception of the values of the
687 `xrd:Query`, `xrd:Expires`, and `xrd:XRD/xrd:LocalID` elements).

688 If a parent authority has assigned a child authority a persistent identifier along with one or more
689 reassignable identifiers, the parent authority SHOULD return the persistent identifier as a
690 `xrd:XRD/xrd:LocalID` value in any XRD returned for a reassignable identifier. The reverse
691 MAY also be true, however parent authorities MAY adopt privacy or other policies that restrict the
692 reassignable synonyms returned for any particular resolution request.

693 A parent authority SHOULD NOT permit a child authority to edit a `LocalID` value in an XRD
694 without authenticating the child authority and verifying that the child authority is authorized to use
695 this `LocalID` value.

696 At the service level, the `xrd:XRD/xrd:Service/xrd:LocalID` element MAY be used to
697 express either a relative or absolute identifier for the child authority in the context of the specific
698 service being described. If present, consuming applications SHOULD use the value of the highest
699 priority instance of the `xrd:XRD/xrd:Service/xrd:LocalID` element to identify the child
700 authority in the context of this service endpoint. If not present, consuming applications SHOULD
701 select an identifier follow the synonym selection rules specified in section 5.3.

702 5.2.2 CanonicalID

703 In XRD documents, a hierarchical canonical synonym is expressed using the `xrd:CanonicalID`
704 element. From the perspective of the parent authority providing the XRD, this element represents
705 the recommended hierarchical global synonym for the child authority (and thus has a cardinality of
706 zero-or-one).

707 Parent authorities SHOULD ALWAYS publish a CanonicalID element in an XRD, even if the
708 value is identical to the QXRI, as this: a) makes it unambiguous to consuming applications the
709 hierarchical global synonym they should use to identify the target resource in the context of the
710 parent authority, and b) enables CanonicalEquivID verification (below). Once assigned, a parent
711 authority SHOULD NEVER change a CanonicalID value or stop asserting a CanonicalID element
712 in an XRD that describes the target resource.

713 A CanonicalID MUST be an absolute identifier. For durability of the reference, a CanonicalID
714 SHOULD be a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141]. If
715 a persistent CanonicalID value is present, applications SHOULD use it as a persistent identifier
716 for the target resource and treat all other reassignable identifiers as temporary synonyms.

717 Like a LocalID, resolution of a CanonicalID MUST return the same XRD as the XRD in which it
718 appears (with the exception of the values of the `xrd:Query`, `xrd:Expires`, and
719 `xrd:XRD/xrd:LocalID` elements).

720 A CanonicalID synonym SHOULD NOT be trusted unless it is verified. This can be done
721 automatically during resolution by an XRI resolver; see section 12. A parent authority SHOULD
722 NOT permit a child authority to edit the CanonicalID value in an XRD without authenticating the
723 child authority and verifying that the child authority is authorized to use this CanonicalID value.

724 5.2.3 EquivID

725 In XRD documents, a polyarchical global synonym is asserted using the `xrd:EquivID` element.
726 This may be used to assert:

- 727 • Other hierarchical global synonyms for the target resource besides the CanonicalID. The final
728 local synonym in all such identifiers MUST be assigned by the parent authority and SHOULD
729 match a LocalID asserted in the same XRD.
- 730 • Polyarchical global synonyms for the target resource that are NOT assigned by the parent
731 authority.

732 An EquivID MUST be an absolute identifier. For durability of the reference, it is RECOMMENDED
733 to use a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

734 If an EquivID will be used to establish equivalence of identity across trust domains, it SHOULD be
735 verified. This function is not performed automatically by XRI resolvers during resolution but may
736 be easily performed by consuming applications using one additional XRI resolution call. See
737 section 13.5. A parent authority SHOULD NOT permit a child authority to edit the EquivID value in
738 an XRD without authenticating the child authority and verifying that the child authority is
739 authorized to use this EquivID value.

740 5.2.4 CanonicalEquivID

741 In XRD documents, a polyarchical canonical synonym is asserted using the
742 `xrd:CanonicalEquivID` element. From the perspective of the parent authority providing the
743 XRD, this element represents the recommended polyarchical global synonym for the child
744 authority (and thus has a cardinality of zero-or-one).

745 Parent authorities MUST NOT publish an CanonicalEquivID element in an XRD if: a) there is no
746 CanonicalID element, or b) the value of the CanonicalEquivID element is equivalent to the value
747 of the CanonicalID element. Again, these rules makes it unambiguous to consuming applications
748 that there is a different polyarchical global synonym used to identify the target resource.

749 Consuming applications SHOULD map the CanonicalEquivID to the CanonicalID and consider
750 both canonical identifiers for the target resource in their respective contexts.

751 Unlike the CanonicalID, the CanonicalEquivID asserted by a parent authority MAY change over
752 time if the preferred context for global identification of the target resource changes.

753 A CanonicalEquivID MUST be an absolute identifier. For durability of the reference, a
754 CanonicalEquivID SHOULD be a persistent identifier such as a persistent XRI [XRISyntax] or a
755 URN [RFC2141].

756 Unlike a CanonicalID, resolution of an CanonicalEquivID SHOULD return a different XRD than
757 the XRD in which it appears.

758 A CanonicalEquivID synonym SHOULD NOT be trusted unless it is verified. This can be done
759 automatically during resolution by an XRI resolver; see section 12. A parent authority SHOULD
760 NOT permit a child authority to use a hierarchical global synonym as a CanonicalEquivID value in
761 an XRD without authenticating the child authority and verifying that the child authority is
762 authorized to use this hierarchical CanonicalEquivID value. However parent authorities MAY
763 permit child authorities to use their choice of a polyarchical CanonicalEquivID value as verification
764 of such a value is outside the scope of the parent authority. See section 12.

765 5.2.5 Ref

766 The `xrd:Ref` element plays a special role in XRDS architecture. The rules for its usage are
767 identical to those for the `EquivID` element with one exception: it MUST NOT be used to assert a
768 synonymous identifier for the target resource but only to assert another XRD that describes the
769 target resource. As with `CanonicalEquivID`, resolution of an `Ref` SHOULD return a different XRD
770 than the XRD in which it appears.

771 Like a `LocalID`, a `Ref` can be used at both the authority level of an XRD (as a child of the root
772 `xrd:XRD` element) and the service level (as a child of the root `xrd:XRD/xrd:Service`
773 element). The complete rules for `Ref` processing in XRI resolution are specified in section 12.

774 At the authority level, a `Ref` is an assertion that another parent authority is authorized to describe
775 the child authority with an XRD, and that a resolver MUST consider the XRDs from both
776 authorities as a "logical union". See section 12.1 for authority reference processing rules.

777 At the service level, a service `Ref` MUST be an absolute HTTP(S) URI that directly references
778 another XRDS document describing the target authority in the context of a specific service. See
779 section 12.2 for service reference processing rules.

780 Like `LocalID`, a `Ref` is asserted directly by the parent authority and requires no verification.

781 5.3 Synonym Selection Rules

782 If an XRD contains one or more synonym elements and a consuming application must select a
783 canonical synonym to a reference a target resource, it SHOULD apply the following rules:

- 784 1. Select the value of the `xrd:XRD/xrd:CanonicalEquivID` element if present and
785 verified as specified in section 13.
- 786 2. Select the value of the `xrd:XRD/xrd:CanonicalID` element if present and verified as
787 specified in section 13.
- 788 3. Use the query identifier, which is always present and does not require verification.

789

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

790

791

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

792

793

794

Comment [DSR12]: Do we need to reference?

795

6.1 Overview

796

There are two protocols for discovery of an XRDS document from an HTTP(S) URI:

797

1. *HEAD protocol*: using an HTTP(S) HEAD request to obtain a header with XRDS document location information as specified in section 6.2.

798

799

2. *GET protocol*: using an HTTP(S) GET request with content negotiation as specified in section 6.3.

800

801

An XRDS server MUST support the GET protocol and MAY support the HEAD protocol. An XRDS client MAY attempt the HEAD protocol but MUST attempt the GET protocol if the HEAD protocol fails.

802

803

Comment [DSR13]: This overview section was rewritten in ED04. Please review.

804

6.2 HEAD Protocol

805

Under this protocol the XRDS client MUST begin by issuing an HTTP(S) HEAD request. This request SHOULD include an Accept header specifying the content type `application/xrds+xml`.

806

807

808

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

809

810

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

811

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

812

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 XRDS client MUST then request this document as specified in section 6.3.

813

814

815

If the response includes the second option above, the XRDS client MUST request the XRDS document from the original HTTP(S) URI as specified in section 6.3.

816

817

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

818

819

If response includes neither of the two options above, this protocol fails and the XRDS client MUST fall back to using the protocol specified in section 6.3.

820

821

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

822

6.3 GET Protocol

823

Under this protocol the XRDS client MUST begin by issuing an HTTP(S) GET request. This request SHOULD include an Accept header specifying the content type `application/xrds+xml`.

824

825

826

The XRDS server response MUST be one of four options:

827

1. HTTP(S) response-headers only as defined in section 6.2.

- 828 2. HTTP(S) response-headers as defined in section 6.2 together with a document, which
829 MAY be either document type specified in options 3 or 4 below.
- 830 3. A valid HTML document with a <head> element that includes a <meta> element with an
831 http-equiv attribute equal to X-XRDS-Location.
- 832 4. A valid XRDS document (content type application/xrds+xml).

833 If the response is only HTTP(S) response headers as defined in section 6.2, or if it includes any
834 document other than the two document types defined in the third and fourth above, the protocol
835 MUST proceed as defined in section 6.2, except that there is no fallback to this section if that
836 protocol fails.

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

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

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

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

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

852

7 XRI Resolution Inputs and Outputs

853
854
855
856
857
858

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 10. For purposes of illustration, a binding to a non-normative, language-neutral API is suggested in Appendix C.

859

7.1 Inputs

860
861
862
863
864

Table 8 summarizes the logical input parameters to XRI resolution and whether they are applicable in the authority resolution phase (sections 8 and 9) or the service endpoint selection phase (section 11). 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	Section
QXRI (query XRI) including Authority String, Path String, and Query String	xs:anyURI	Required	N/A	Authority resolution	7.1.1
Resolution Output Format	xs:string (media type)	Optional	Null	Authority resolution	7.1.2
Service Type	xs:anyURI	Optional	Null	Service endpoint selection	7.2.3
Service Media Type	xs:string (media type)	Optional	Null	Service endpoint selection	7.1.4

865

Table 8: Input parameters for XRI resolution.

866
867

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

868
869
870
871

1. The presence of an 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.

872
873

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

874 **7.1.1 QXRI (Authority String, Path String, and Query String)**

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

Logical Parameter Name	Type	Required/Optional	Value
Authority String	xs:string	Required	Contents of the authority segment of the QXRI, not including 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.

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

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

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

- 882 1. It **MUST** be a valid absolute XRI according to the ABNF defined in [XRISyntax]. To
883 resolve a relative XRI reference, it must be converted into an absolute XRI using the
884 procedure defined in section 2.4 of [XRISyntax].
- 885 2. For authority or proxy resolution as defined in this specification, the QXRI **MUST** be in
886 URI-normal form as defined in section 2.3.1 of [XRISyntax]. A local resolver API **MAY**
887 support the input of other XRI forms but **SHOULD** document the normal form(s) it
888 supports and its normalization policies.
- 889 3. When a QXRI is included as part of an HXRI (section 10.2) for XRI proxy resolution, the
890 QXRI **MUST** be normalized as specified in section 10.2, and all HXRI query parameters
891 **MUST** follow the encoding rules specified in section 10.3.

892 **7.1.2 Resolution Output Format**

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

- 894 • The media type for the resolution response.
- 895 • Whether generic or trusted resolution must be used by the resolver.
- 896 • Whether references should be followed during resolution.
- 897 • Whether CanonicalID verification should be performed during resolution.
- 898 • Whether service endpoint selection should be performed on the final XRD.
- 899 • Whether default matches should be ignored during service endpoint selection.

- 900 Following are the normative requirements for the use of this parameter.
- 901 1. The value of Resolution Output Format MUST be one of the values specified in Table 5
902 and MAY include any of the media type parameters specified in Table 6.
 - 903 2. If the value of the `https` media type parameter is `true`, the resolver MUST use the
904 HTTPS trusted authority resolution protocol specified in section 9.1 (or return an error
905 indicating this is not supported).
 - 906 3. If the value of the `saml` media type parameter is `true`, the resolver MUST use the SAML
907 trusted authority resolution protocol specified in section 9.2 (or return an error indicating
908 this is not supported).
 - 909 4. If the value of both the `https` and `saml` media type parameters is `true`, the resolver
910 MUST use the HTTPS+SAML trusted authority resolution protocol specified in section 9.3
911 (or return an error indicating this is not supported).
 - 912 5. If the value of the `cid` media type parameter is `true`, the resolver MUST perform
913 CanonicalID verification as specified in section 12.
 - 914 6. If the value of the `cid` media type parameter is `false` or null, or if the parameter is
915 absent, the resolver MUST NOT perform CanonicalID verification.
 - 916 7. If the value of the `refs` media type parameter is `true` or null, or if the parameter is
917 absent, the resolver MUST perform reference processing as defined in section 12 if it is
918 necessary to complete resolution (or return an error indicating this is not supported).
 - 919 8. If the value of the `refs` media type parameter is `false`, the resolver MUST NOT
920 perform reference processing during resolution.
 - 921 9. If the value of the `sep` media type parameter is `true`, the resolver MUST perform service
922 endpoint selection on the final XRD (or return an error indicating this is not supported).
 - 923 10. If the value of the `sep` media type parameter is `false` or null, or if the parameter is
924 absent, the resolver MUST NOT perform service endpoint selection on the final XRD
925 unless it is required to produce a URI List or HTTP(S) redirect. See section 7.2.
 - 926 11. If the value of the `ndefault` media type parameter is `type`, `path`, or `mediatype`,
927 the resolver MUST ignore default service endpoint selection element matches as
928 specified in section 11.3.2.

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

931 7.1.3 Service Type

932 The Service Type is an OPTIONAL value of type `xs:anyURI` used to request a specific type of
933 service in the service endpoint selection phase (section 10). The value of this parameter MUST
934 be a valid absolute XRI, IRI, or URI in URI-normal form as defined by **[XRISyntax]**. (Note that
935 URI-normal form is specified so that this parameter may be passed to a proxy resolver in a QXRI
936 query parameter as defined in section 10.) The Service Type values defined for XRI resolution
937 services are specified in section 3.1.2.

938 7.1.4 Service Media Type

939 The Service Media Type is an OPTIONAL string used to request a specific media type in the
940 service endpoint selection phase (section 10). The value of this parameter MUST be a valid
941 media type as defined by **[RFC2046]**. The Service Media Type values defined for XRI resolution
942 services are specified in section 3.3.

943 7.2 Outputs

944 Table 10 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 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>

945 *Table 10: Outputs of XRI resolution.*

946 The following sections provide additional construction and validation requirements.

947 7.2.1 XRDS Document

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

- 950 1. The output MUST be a valid XRDS document according to the schema defined in
951 Appendix A. Also, any nested XRDS documents included as a result of reference
952 processing (section 12) must also be valid.
- 953 2. Each of the contained XRD elements must be a valid XRD document according to the
954 schema defined in Appendix A.
- 955 3. The XRD elements MUST conform to the additional requirements in section 4.
- 956 4. If the value of the `saml` parameter of the Resolution Output Format is `true`, the XRD
957 element MUST further conform to the additional requirements in section 9.2.
- 958 5. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST
959 be performed as defined in section 11, even if the values of all three service endpoint
960 selection input parameters (Service Type, Service Media Type, and Path String) are null.
961 This ensures that a resolver will perform reference processing (section 12) to locate a
962 requested service endpoint (or a default service endpoint) if necessary. **IMPORTANT:**
963 Regardless of whether reference processing is performed or not, no filtering of the final
964 XRD is performed when returning an XRDS document. Filtering is only performed when
965 the requested Resolution Output Format is an XRD document – see the next section.
- 966 6. If reference processing is necessary during the authority resolution or service endpoint
967 selection process, it MUST result in a nested XRDS document as defined in section
968 12.1.2. Again, no filtering of the final XRD is performed when returning an XRDS
969 document.
- 970 7. If the output is an error, this error MUST be returned using the `xrd:Status` element of
971 the final XRD in the XRDS document as defined in section 14.

972 7.2.2 XRD Document

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

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

- 977 2. The XRD elements MUST conform to the additional requirements in section 4.
- 978 3. [TODO – add processing instructions for the case where the `saml` media type parameter
979 is `true`.]
- 980 4. If the value of the `sep` media type parameter is `false` or null, or if this parameter is
981 absent, the XRD MUST be the final XRD in the XRDS document produced as a result of
982 authority resolution. Service endpoint selection or any other filtering of the XRD
983 document MUST NOT be performed.
- 984 5. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST
985 be performed as defined in section 11, even if the values of all three service endpoint
986 selection input parameters (Service Type, Service Media Type, and Path String) are null.
- 987 6. If service endpoint selection is performed, the only `xrd:Service` elements in the XRD
988 document MUST be those selected according to the rules specified in section 11. If no
989 service endpoints were selected by those rules, no `xrd:Service` elements will be
990 present. In addition, all elements in the XRD document that are subject to the global
991 `xrd:priority` attribute (even if the attribute is absent or null) MUST be returned in
992 order of highest to lowest priority as defined in section 4.3.3. Any other filtering of the
993 XRD document MUST NOT be performed. Note that this means that if the XRD
994 document includes a SAML signature element as defined in section 9.2, this element is
995 still returned in the XRD document even though it may not be able to be verified by a
996 consuming application.
- 997 7. If the output is an error, this error MUST be returned using the `xrd:Status` element as
998 defined in section 14.

999 7.2.3 URI List

1000 If the value of the Resolution Output Format parameter is `text/uri-list`, the following rules
1001 apply.

- 1002 1. For this output, service endpoint selection is REQUIRED.
- 1003 2. If authority resolution and service endpoint selection are both successful, the output
1004 MUST be a valid URI List as defined by section 5 of [RFC2483].
- 1005 3. If, after applying the service endpoint selection rules, more than one service endpoint is
1006 selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as
1007 defined in section 4.3.3.
- 1008 4. If the final selected `xrd:XRD/xrd:Service` element does not contain an
1009 `xrd:XRD/xrd:Service/xrd:URI` element but does contain at least one
1010 `xrd:XRD/xrd:Service/xrd:Ref` element, service ref processing MUST be
1011 performed as described in section 12.2.
- 1012 5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)
1013 MUST be constructed as defined in section 11.6.
- 1014 6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`
1015 elements within the selected `xrd:Service` element as defined in section 4.3.3. When
1016 two or more of the source `xrd:URI` elements have equal priority, their constructed URIs
1017 SHOULD be returned in random order. Any other filtering of the URI list MUST NOT be
1018 performed.
- 1019 7. If the output is an error, it MUST be returned with the content type `text/plain` as
1020 defined in section 14.

1021 7.2.4 HTTP(S) Redirect

1022 In XRI proxy resolution, the Resolution Output Format parameter may be null. In this case the
1023 output of a proxy resolver is an HTTP(S) redirect as defined in section 10.6.

1024 8 Generic Authority Resolution

1025 Authority resolution is the first phase of XRI resolution as described in section 1.1. It applies only
1026 to the Authority String of the QXRI. This may be either an *XRI authority* or an *IRI authority* as
1027 described in section 2.2.1 of [XRISyntax].

1028 XRI authorities and IRI authorities have different syntactic structures, partially due to the higher
1029 level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved
1030 to XRDS documents one subsegment at a time as specified in section 8.1. IRI authorities, since
1031 they are based on DNS names or IP addresses, are resolved into an XRDS document through a
1032 special HTTP(S) request using the entire IRI authority segment as specified in section 8.2.

1033 8.1 XRI Authority Resolution

1034 8.1.1 Service Type and Service Media Type

1035 The protocol defined in this section is identified by the values in Table 11.

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

1036 *Table 11: Service Type and Service Media Type values for generic authority resolution.*

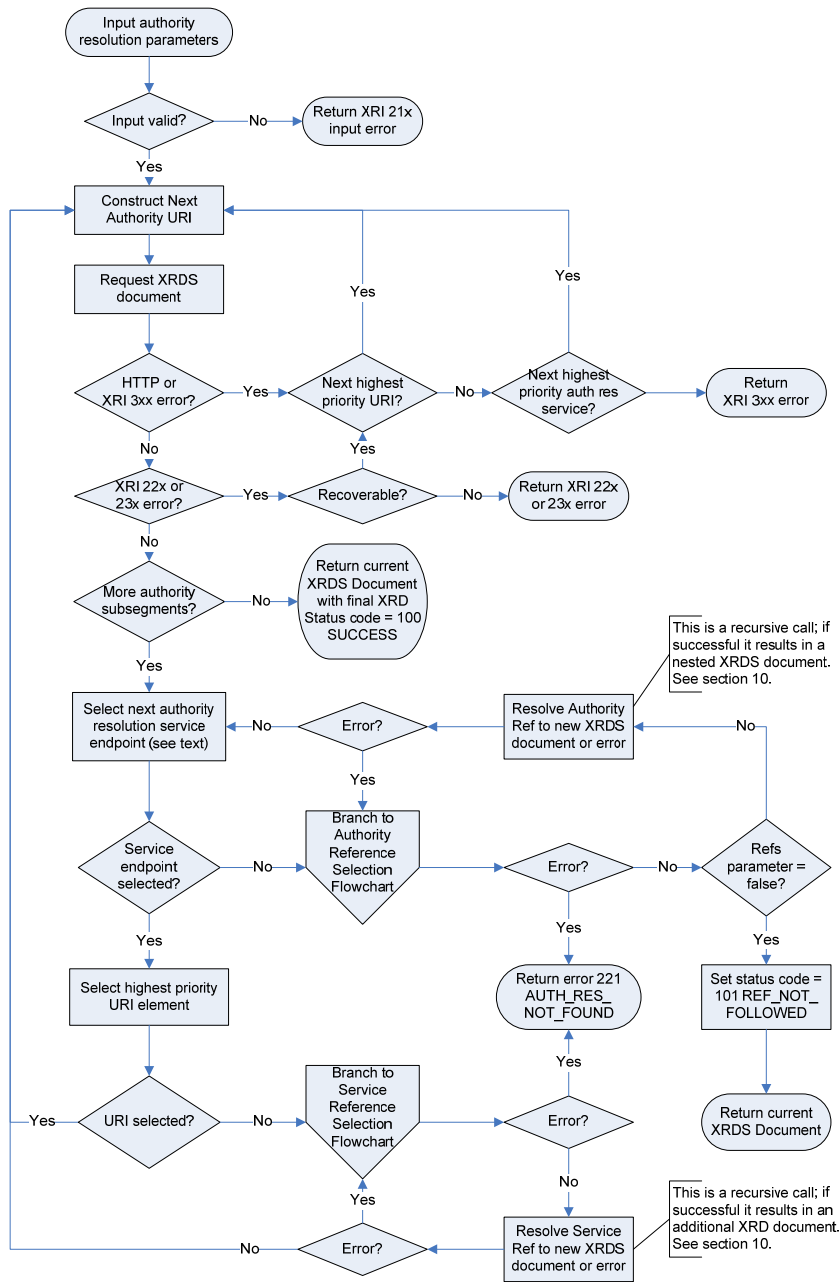
1037 A generic authority resolution service endpoint advertised in an XRDS document MUST use the
1038 Service Type identifier and MAY use the Service Media Type identifier defined in Table 11.

1039 **BACKWARDS COMPATABILITY NOTE:** Earlier drafts of this specification used a media type
1040 parameter called *trust*. This has been deprecated in favor of new parameters for each trusted
1041 resolution option, i.e., `https="true"` and `saml="true"`. However implementations SHOULD
1042 consider the following values equivalent both for the purpose of service endpoint selection within
1043 XRDS documents and as HTTP(S) Accept header values in XRI authority resolution requests:

```
1044 application/xrds+xml  
1045 application/xrds+xml;trust=none  
1046 application/xrds+xml;https=false  
1047 application/xrds+xml;saml=false  
1048 application/xrds+xml;https=false;saml=false  
1049 application/xrds+xml;saml=false;https=false
```

1050 **8.1.2 Protocol**

1051 Figure 6 (non-normative) shows the overall logical flow of generic authority resolution. Note that it
 1052 does not include the additional steps involved in service endpoint selection (section 11),
 1053 reference processing (section 12), or CanonicalID verification (section 13).



1054

1055 *Figure 6: Authority resolution flowchart*

1056 Following are the normative requirements for behavior of an XRI resolver and an XRI resolution
1057 service when performing generic XRI authority resolution:

- 1058 1. The resolver MUST be preconfigured with the XRDS document describing the community
1059 root authority for the XRI to be resolved as defined in section 8.1.3, or have another
1060 equivalent means of obtaining this metadata. The resolver MAY obtain this using a self-
1061 describing XRDS document as defined in section 8.1.4.
- 1062 2. Resolution of each subsegment in the Authority String after the community root
1063 subsegment MUST proceed in subsegment order (left-to-right) using fully qualified
1064 subsegment values as defined in section 8.1.5. If the final output is an XRDS document,
1065 this document MUST contain an ordered list of `xrd:XRD` elements—one for each
1066 authority subsegment successfully resolved by the resolver client. This list MUST appear
1067 in the same order as the corresponding subsegments in the Authority String. In addition,
1068 any authority references followed MUST be represented by nested `xrd:XRDS`
1069 documents as defined in section 12.1.2 and any service references followed MUST be
1070 represented by additional `xrd:XRD` elements as defined in section 12.2.2.
- 1071 3. Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as
1072 defined in section 8.1.6.
- 1073 4. The resolver MAY request that a recursing authority resolution service perform resolution
1074 of multiple subsegments as defined in section 8.1.7.
- 1075 5. For each iteration of the authority resolution process, the next authority resolution service
1076 endpoint MUST be selected as specified in section 8.1.8 and an HTTP(S) URI (called the
1077 Next Authority URI) MUST be constructed as specified in section 8.1.8.
- 1078 6. Each resolution request MUST be an HTTP(S) GET to the Next Authority URI and MUST
1079 contain an Accept header with the media type identifier defined in Table 11. Note that in
1080 XRI authority resolution, this Accept header is NOT interpreted as an XRI resolution input
1081 parameter, but simply as the media type being requested from the server. This differs
1082 from XRI proxy resolution, where the Accept header MAY be used to specify the Service
1083 Media Type resolution parameter. See section 10.4.
- 1084 7. The ultimate HTTP(S) response from an authority resolution service to a successful
1085 resolution request MUST contain either: a) a 2XX response with a valid XRDS document
1086 containing an XRD element for each authority subsegment resolved, or b) a 304
1087 response signifying that the cached version on the resolver is still valid (depending on the
1088 client's HTTP(S) request). There is no restriction on intermediate redirects (i.e., 3XX
1089 result codes) or other result codes (e.g., a 100 HTTP response) that eventually result in a
1090 2XX or 304 response through normal operation of [RFC2616].
- 1091 8. The HTTP(S) response from an authority resolution service MUST return the media type
1092 requested by the resolver. The response SHOULD NOT include any media type
1093 parameters supplied by the resolver in the request. If the resolver receives such
1094 parameters in the response, the resolver MUST ignore them and do its own independent
1095 verification that the response fulfills the requested parameters.
- 1096 9. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in
1097 the resolution process and the resolver SHOULD return the appropriate error code and
1098 context message as specified in section 14. In recursing resolution, such an error MUST
1099 be returned by the recursing authority resolution service to the resolver as specified in
1100 section 14.3.
- 1101 10. If an XRD does not include the next requested authority service endpoint but includes
1102 one or more `xrd:Ref` elements, the resolver MUST perform reference processing as
1103 defined in section 12 unless the `refs` media type parameter defined in Table 6 is set to
1104 `false`. If the `refs` media type parameter is set to `false` and the XRD contains at least
1105 one `xrd:Ref` element that could be followed, the resolver MUST return a response with

1106 a status code of 101 REF_NOT_FOLLOWED. (Note that such reference processing, if
1107 successful, will result in a separate nested XRDS document describing the resolved
1108 reference as defined in section 12.1.2.)

1109 11. [TODO – add rule for using a service ref.]

← Formatted: Bullets and Numbering

1110 12. A successful response that does not include the next required authority service endpoint
1111 in the XRD and does not include any `xrd:Ref` elements MUST return an error with a
1112 status code of 221 AUTH_RES_NOT_FOUND regardless of the value of the `refs` media
1113 type parameter.

1114 13. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section
1115 15. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent
1116 possible to maintain the efficiency and scalability of the HTTP-based resolution system.
1117 The recommended use of HTTP caching headers is described in more detail in section
1118 15.2.1.

1119 8.1.3 Community Root Authorities

1120 Identifier management policies are defined on a community-by-community basis. For XRI
1121 authorities, the resolution community is specified by the first (leftmost) subsegment of the
1122 authority segment of the XRI. This is referred to as the *community root authority*. When a
1123 resolution community chooses to create a new community root authority, it SHOULD define
1124 policies for assigning and managing identifiers under this authority. Furthermore, it SHOULD
1125 define what resolution protocol(s) may be used for these identifiers.

1126 For an XRI authority, the community root may be either a global context symbol (GCS) character
1127 or top-level cross-reference as specified in section 2.2.1.1 of [XRISyntax]. In either case, the
1128 corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution
1129 service endpoints for that community.

1130 The community root authority SHOULD publish a self-describing XRDS document as defined in
1131 section 8.1.4. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as
1132 the community's root authority resolution service endpoints. This community root XRDS
1133 document, or its location, must be known *a priori* and is part of the configuration of an XRI
1134 resolver, similar to the specification of root DNS servers for a DNS resolver. Note that is not
1135 strictly necessary to publish this information in an XRDS document—it may be supplied in any
1136 format that enables configuration of the XRI resolvers in the community. However publishing a
1137 self-describing XRDS document at a known location simplifies this process and enables dynamic
1138 configuration of community resolvers.

1139 It is also a recommended best practice for a community root XRDS document to contain:

- 1140 • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- 1141 • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML
1142 trusted resolution is supported.
- 1143 • Both of the above if HTTPS+SAML trusted resolution is supported.
- 1144 • The service endpoints and supported media types of the community's XRI proxy resolver(s) if
1145 proxy resolution is supported.

1146 [TODO] For a list of public community root authorities and the locations of their community root
1147 XRDS documents, see the XRI Technical Committee home page at [http://www.oasis-](http://www.oasis-open.org/committees/xri)
1148 [open.org/committees/xri](http://www.oasis-open.org/committees/xri).

Comment [DSR14]: OPEN ISSUE:
remains open for discussion.

1149 8.1.4 Self-Describing XRDS Documents

1150 An identifier authority MAY publish a self-describing XRDS document, i.e., one produced by the
1151 same identifier authority that it describes. A resolver MAY request a self-describing XRDS
1152 document from a target identifier authority using either of two methods:

- 1153 1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution
1154 service endpoint, it may use the resolution protocol specified in section 5 to request an
1155 XRDS document directly from this HTTP(S) URI(s). This HTTP(S) URI may be known a
1156 priori (as is often the case with community root authorities, above), or it may be
1157 discovered from other identifier authorities via the resolution protocols defined in this
1158 specification.
- 1159 2. If the resolver knows: a) an XRI of the target authority as a community root authority, and
1160 b) the location of a proxy resolver configured for this community root authority, it may use
1161 the proxy resolution protocol specified in section 10 to query the proxy resolver for the
1162 community root authority XRI. This query MUST include only a single subsegment
1163 identifying the community root authority and MUST NOT include any additional
1164 subsegments.

1165 An example of the first method, if a identifier authority had an authority resolution service
1166 endpoint at `http://example.com/auth-res-service/`, would be to issue an HTTP(S) GET
1167 request to that URI with an Accept header specifying the content type
1168 `application/xrds+xml`. See section 6.3 for more details.

1169 An example of the second method, if a identifier authority had the community root authority
1170 identifier `xri://(example)` and was registered with the XRI proxy resolver
1171 `http://xri.example.com/`, would be to issue an HTTP(S) GET request to the following URI:

1172 `http://xri.example.com/(example)?_xrd_r=application/xrds+xml`

1173 See section 10 for more details.

1174 Note that a proxy resolver may use the first method to publish its own self-describing XRDS
1175 document at the HTTP(S) URI(s) for its proxy resolution service.

1176 **IMPORTANT:** A self-describing XRDS document MUST only be issued by an identifier authority
1177 when describing itself. It MUST NOT be included when the identifier authority is describing a
1178 different identifier authority. In the latter case the self-describing XRDS document for the
1179 community root authority is implicit. It MAY be explicitly requested by the resolver if needed for
1180 dynamic configuration of the resolver or trust verification of other XRI resolution chains.

1181 8.1.5 Qualified Subsegments

1182 A qualified subsegment is defined by the productions whose names start with “xri-subseg” in
1183 section 2.2.3 of **[XRISyntax]** including the leading syntactic delimiter (“*” or “!”). A qualified
1184 subsegment MUST include the leading syntactic delimiter even if it was optionally omitted in the
1185 original XRI (see section 2.2.3 of **[XRISyntax]**).

1186 If the first subsegment of an XRI authority is a GCS character and the following subsegment does
1187 not begin with a “*” (indicating a reassignable subsegment) or a “!” (indicating a persistent
1188 subsegment), then a “*” is implied and MUST be added when constructing the qualified
1189 subsegment as specified in section 8.1.8. Table 12 and Table 13 illustrate the differences
1190 between parsing a reassignable subsegment following a GCS character and parsing a cross-
1191 reference, respectively.

1192

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

1193 *Table 12: 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

1194 *Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.*

1195 8.1.6 Cross-References

1196 Any subsegment within an XRI authority segment may be a cross-reference (see section 2.2.2 of
 1197 **[XRI Syntax]**). Cross-references are resolved identically to any other subsegment because the
 1198 cross-reference is considered opaque, i.e., the value of the cross-reference (including the
 1199 parentheses) is the literal value of the subsegment for the purpose of resolution.

1200 Table 14 provides several examples of resolving cross-references. In these examples,
 1201 subsegment "lb" resolves to a Next Authority Service Endpoint URI of "http://example.com/xri-
 1202 authority/" and recursing authority resolution is not being requested.
 1203

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)

1204 *Table 14: Examples of the Next Authority URIs constructed using different types of cross-references.*

1205 8.1.7 Recursing Authority Resolution

1206 If an authority resolution service offers recursing resolution, an XRI resolver may request
 1207 resolution of multiple authority subsegments in one transaction. If a resolver makes such a
 1208 request, the responding authority resolution service MAY perform the additional recursing
 1209 resolution steps requested. In this case the recursing authority resolution service acts as a
 1210 resolver to the other authority resolution service endpoints that need to be queried. Alternatively,
 1211 the recursing authority resolution service may retrieve XRDs from its local cache until it reaches a
 1212 subsegment whose XRD is not locally cached, or it may simply recurse only as far as it is

1213 authoritative. If an authority resolution service performs any recursing resolution, it MUST return
1214 an ordered list of `xrd:XRDS` elements (and nested `xrd:XRDS` elements if references are
1215 followed) in an `xrd:XRDS` document for all subsegments resolved as defined in section 8.1.2.
1216 The recursing authority resolution service MAY resolve fewer subsegments than requested by the
1217 resolver. The recursing authority resolution service is under no obligation to resolve more than
1218 the first subsegment (for which it is, by definition, authoritative).
1219 If the recursing authority resolution service does not resolve the entire set of subsegments
1220 requested, the resolver is responsible for continuing the authority resolution process itself. At any
1221 stage, however, the resolver MAY request that the next authority resolution service recursively
1222 resolve any remaining subsegments.

1223 8.1.8 Construction of the Next Authority URI

1224 At each step in authority resolution, a URI must be constructed for the next HTTP(S) request.
1225 This URI is constructed by the XRI resolver from two strings—one representing the next authority
1226 resolution service endpoint selected from the current XRD, and the other representing the next
1227 unresolved subsegment or group of subsegments in the QXRI Authority String.

1228 The process for selecting the next authority resolution service endpoint from the current XRD is
1229 defined in section 11. For generic authority resolution, this selection process MUST use the
1230 parameters specified in Table 11. For trusted authority resolution, this selection process MUST
1231 use the parameters specified in Table 15, Table 16, or Table 17. In all cases, an explicit match on
1232 the `xrd:XRDS/Service/xrd:Type` element is REQUIRED, so during authority resolution, a
1233 resolver MUST set the `nodefault` parameter to a value of `nodefault=type` in order to
1234 override selection of a default service endpoint as specified in section 11.3.2.

1235 From the output of the service endpoint selection process, the resolver MUST select the highest
1236 priority URI of the highest priority authority resolution service endpoint. Next, the resolver MUST
1237 apply the service endpoint URI construction algorithm based the value of the `append` attribute as
1238 defined in section 11.7.

1239 This fully constructed URI is called the *Next Authority Service Endpoint URI*. If this URI does not
1240 end with a forward slash (“/”), one MUST be appended before proceeding.

1241 The second string is called the *Next Authority String* and it consists of either:

- 1242 • The next fully qualified subsegment to be resolved (see section 8.1.5), or
- 1243 • In the case of recursing resolution, the next fully qualified subsegment to be resolved plus
1244 any additional subsegments for which recursing resolution is requested (see section 8.1.6).

1245 The final step is to append the Next Authority String to the path component of the Next Authority
1246 Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

1247 Construction of the Next Authority URI is more formally described in this pseudocode for
1248 resolving a “next-auth-string” via a “next-auth-sep-uri”:

```
1249 if (path portion of next-auth-sep-uri does not end in "/"):  
1250     append "/" to path portion of next-auth-sep-uri  
1251  
1252 if (next-auth-string is not preceded with "*" or "!" delimiter):  
1253     prepend "*" to next-auth-string  
1254  
1255 append uri-escape(next-auth-string) to path of next-auth-sep-uri
```

1256 8.2 IRI Authority Resolution

1257 From the standpoint of generic authority resolution, an IRI authority segment represents either a
1258 DNS name or an IP address at which an XRDS document describing the authority may be
1259 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET
1260 request to a URI constructed from the IRI authority segment. The resulting XRDS document can
1261 then be consumed in the same manner as one obtained using XRI authority resolution.

1262 While the use of IRI authorities provides backwards compatibility with the large installed base of
1263 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of
1264 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities
1265 are not recommended for new deployments of XRI identifiers.

1266 This section defines IRI authority resolution as a simple extension to the XRI authority resolution
1267 protocol defined in the preceding section.

1268 **8.2.1 Service Type and Media Type**

1269 Because IRI authority resolution takes place at a level “below” XRI authority resolution, it cannot
1270 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority
1271 resolution uses the same media type as generic XRI authority resolution.

1272 **8.2.2 Protocol**

1273 Following are the normative requirements for IRI authority resolution that differ from generic XRI
1274 authority resolution:

1275 1. The next authority URI is constructed by extracting the entire IRI authority segment and
1276 prepending the string “http://”. See the exception in section 8.2.3.

1277 2. The HTTP GET request MUST include an HTTP Accept header containing only the
1278 following:

1279 `Accept: application/xrds+xml`

1280 3. The HTTP GET request MUST have a Host: header (as defined in section 14.23 of
1281 **[RFC2616]**) containing the value of the IRI authority segment.

1282 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document
1283 containing the XRD describing that authority.

1284 5. The responding server MUST use the value of the Host header to populate the
1285 `xrd:XRD/xrd:Query` element in the resulting XRD. For example:

1286 `Host: example.com`

1287 Note that because IRI authority resolution is required to process the entire IRI authority segment
1288 in a single step, recursing authority resolution does not apply.

1289 **8.2.3 Optional Use of HTTPS**

1290 Section 9 of this specification defines trusted resolution only for XRI authorities. Trusted
1291 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to
1292 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY
1293 use HTTPS as the access protocol for retrieving the authority’s XRD. If the resolver is satisfied,
1294 via transport level security mechanisms, that the response is from the expected IRI authority, the
1295 resolver may consider this an HTTPS trusted resolution response as defined in section 9.1.

1296

9 Trusted Authority Resolution

1297
1298
1299

This section defines three options for performing trusted XRI authority resolution as an extension of the generic XRI authority resolution service defined in section 8.1—one using HTTPS, one using SAML assertions, and one using both.

1300

9.1 HTTPS

1301
1302
1303
1304
1305

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

1306

9.1.1 Service Type and Service Media Type

1307

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	https=true

1308

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

1309
1310
1311
1312

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

1313

9.1.2 Protocol

1314
1315

Following are the normative requirements for HTTPS trusted authority resolution that differ from generic XRI authority resolution (section 8.1):

1316
1317
1318
1319
1320
1321
1322
1323
1324
1325

1. All authority resolution service endpoints MUST be selected using the values defined in Table 15.
2. All authority resolution requests including the starting request to a community root authority MUST use the HTTPS protocol as defined in [RFC2818]. A successful HTTPS response MUST be received from each authority in the resolution chain or the resolver MUST output an error.
3. All authority resolution requests MUST contain an HTTPS Accept header with the media type identifier defined in Table 15 (including the `https="true"` parameter).
4. If the resolver finds that an authority in the resolution chain does not support HTTPS, the resolver MUST return a 23x error as defined in section 14.

1326

9.2 SAML

1327
1328
1329
1330
1331
1332

In SAML trusted resolution, the resolver requests a content type of `application/xrds+xml; saml=true` and the authority resolution service responds with an XRDS document containing an XRD with an additional element—a digitally signed SAML [SAML] assertion that asserts the validity of the containing XRD. SAML trusted resolution provides message integrity but does not provide confidentiality. The latter may be achieved by combining it with HTTPS as defined in section 9.3. Message confidentiality may also be achieved with other security protocols used in

1333 conjunction with this specification. SAML trusted resolution also does not provide a means for an
1334 authority to provide different responses for different requestors; client authentication is explicitly
1335 out-of-scope for version 2.0 of XRI resolution.

1336 9.2.1 Service Type and Service Media Type

1337 The protocol defined in this section is identified by the values in Table 16.

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

1338 *Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.*

1339 A SAML trusted resolution service endpoint advertised in an XRD document MUST use the
1340 Service Type identifier and Service Media Type identifier defined in Table 16 (including the
1341 `saml=true` parameter). In addition, for transport security the identifier authority SHOULD offer at
1342 least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1343 9.2.2 Protocol

1344 9.2.2.1 Client Requirements

1345 For a resolver, trusted resolution is identical to the generic resolution protocol (section 8.1) with
1346 the addition of the following requirements:

- 1347 1. All authority resolution service endpoints MUST be selected using the values defined in
1348 Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an
1349 authority unless the authority advertises a resolution service endpoint matching these
1350 values.
- 1351 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is
1352 recommended for confidentiality.
- 1353 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the
1354 media type identifier defined in Table 16 (including the `saml=true` parameter). This
1355 MUST be interpreted as the value of the Resolution Output Format input parameter.
1356 (Clients willing to accept either generic or trusted responses may use a combination of
1357 media type identifiers in the Accept header as described in section 14.1 of [RFC2616].
1358 Media type identifiers SHOULD be ordered according to the client's preference for the
1359 media type of the response. If a client performing generic authority resolution receives an
1360 XRD containing SAML elements, it is NOT REQUIRED to validate the signature or
1361 perform any processing of these elements.)
- 1362 4. A resolver MAY request recursing authority resolution of multiple subsegments as
1363 defined in section 9.2.3.
- 1364 5. The resolver MUST individually validate each XRD in the resolution chain according to
1365 the rules defined in section 9.2.4. When `xrd:XRD` elements come both from freshly-
1366 retrieved XRD documents and from a local cache, a resolver MUST ensure that these
1367 requirements are satisfied each time a resolution request is performed.

1368 9.2.2.2 Server Requirements

1369 For an authority resolution service, trusted resolution is identical to the generic resolution protocol
1370 (section 8.1) with the addition of the following requirements:

- 1371 1. The HTTP(S) response to a trusted resolution request MUST include a content type of
1372 "application/xrds+xml;trust=saml".

- 1373 2. The XRDS document returned by the resolution service MUST contain a
 1374 `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid
 1375 per the processing rules described by [SAML].
- 1376 3. The `saml:Assertion` element MUST contain a valid enveloped digital signature as
 1377 defined by [XMLDSig] and as constrained by section 5.4 of [SAML].
- 1378 4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML
 1379 assertion. Specifically, the signature MUST contain a single
 1380 `ds:SignedInfo/ds:Reference` element, and the URI attribute of this reference
 1381 MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML
 1382 assertion. The URI reference MUST NOT be empty and it MUST refer to the identifier
 1383 contained in the `xrd:XRD/@xml:id` attribute.
- 1384 5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain
 1385 a `ds:KeyInfo` element. If this element is included, it MUST describe the key used to
 1386 verify the digital signature element. However, because the signing key is known in
 1387 advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the
 1388 `ds:Signature` element of the SAML assertion.
- 1389 6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST
 1390 match the XRI authority subsegment requested by the client.
- 1391 7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match
 1392 the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD
 1393 advertising availability of trusted resolution service from this authority as required in
 1394 section 9.2.5.
- 1395 8. The `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
 1396 present and equal to the `xrd:XRD/xrd:Query` element.
- 1397 9. The `NameQualifier` attribute of the
 1398 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
 1399 present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.
- 1400 10. There MUST be exactly one `saml:AttributeStatement` present in the
 1401 `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute`
 1402 element with a `Name` attribute of "`xri://$xrd*(v*2.0)`". This `saml:Attribute`
 1403 element MUST contain exactly one `saml:AttributeValue` element whose text value
 1404 is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that is the immediate
 1405 parent of the `saml:Assertion` element.

Deleted: SAML assertion

Deleted: In

Deleted: ,

Formatted: Font: Not Bold

Deleted: Because

Deleted: digital signature

Comment [DSR15]: TODO: confirm the Name attribute value (Peter)

1406 9.2.3 Recursing Authority Resolution

1407 If a resolver requests trusted resolution of multiple authority subsegments (see section 8.1.6), a
 1408 recursing authority resolution service SHOULD attempt to perform trusted resolution on behalf of
 1409 the resolver as described in this section. However if the resolution service is not able to obtain
 1410 trusted XRDS for one or more additional recursing subsegments, it SHOULD return only the
 1411 trusted XRDS it has obtained and allow the resolver to continue.

1412 9.2.4 Client Validation of XRDS

1413 For each XRD returned as part of a trusted resolution request, the resolver MUST validate the
 1414 XRD according to the rules defined in this section.

- 1415 1. The `xrd:XRD/saml:Assertion` element MUST be present.
- 1416 2. This assertion MUST valid per the processing rules described by [SAML].
- 1417 3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by
 1418 [XMLDSig] and constrained by Section 5.4 of [SAML].

- 1419 4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML
1420 assertion. Specifically, the signature MUST contain a single
1421 `ds:SignedInfo/ds:Reference` element, and the URI attribute of this reference
1422 MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent
1423 of the signed SAML assertion.
- 1424 5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo`
1425 element, the resolver MAY reject the signature if this key does not match the signer's
1426 expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor
1427 that was used to describe the current authority. See section 9.2.5.
- 1428 6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose
1429 resolution resulted in the current XRD.
- 1430 7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1431 `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability
1432 of trusted resolution service from this authority as required in section 9.2.5.
- 1433 8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1434 `NameQualifier` attribute of the
1435 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1436 9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the
1437 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1438 10. There MUST exist exactly one
1439 `xrd:XRD/saml:Assertion/saml:AttributeStatment` with exactly one
1440 `saml:Attribute` element that has a `Name` attribute of "`xri://$xrd*($v*2.0)`". This
1441 `saml:Attribute` element must have exactly one `saml:AttributeValue` element
1442 whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that
1443 is the immediate parent of the signed SAML assertion.

1444 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result
1445 MUST NOT be considered a valid trusted resolution response as defined by this specification.
1446 Note that this does not preclude a resolver from considering alternative resolution paths. For
1447 example, if an XRD advertising SAML trusted resolution service has two or more
1448 `xrd:XRD/xrd:Service/xrd:URI` elements and the response from one service endpoint fails
1449 to meet the requirements above, the client MAY repeat the validation process using the second
1450 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as
1451 defined by this document and SAML trusted resolution may continue.

1452 9.2.5 Correlation of ProviderID and KeyInfo Elements

1453 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at
1454 least one unique persistent service provider identifier expressed in the
1455 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority
1456 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI
1457 authority. A ProviderID may be any valid URI that meets these requirements of persistence and
1458 uniqueness. Examples of appropriate URIs include URNs as defined by [RFC2141] and fully
1459 persistent XRIs expressed in URI-normal form as defined by [XRISyntax].

1460 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in
1461 an XRD advertising trusted authority resolution service with the response received from a trusted
1462 resolution service endpoint. If the signed XRD response contains the same ProviderID as the
1463 XRD used to advertise a service, and the resolver has reason to trust the signature, the resolver
1464 can trust that the XRD response has not been maliciously replaced with another XRD.

1465 There is no defined discovery process for the ProviderID for a community root authority; it must
1466 be published in the community root XRDS document (or other equivalent description—see
1467 section 8.1.3) and verified independently. Once the community root XRDS document is known,

1468 the ProviderID for delegated XRI authorities within this community MAY be discovered using the
1469 `xrd:XRD/xrd:Service/xrd:ProviderID` element of authority resolution service endpoints.
1470 This trust mechanism may also be used for other services offered by an authority.

1471 In addition, the metadata necessary for SAML trusted authority resolution or other SAML [SAML]
1472 interactions MAY be discovered using the `ds:KeyInfo` element (section 4.2.) Again, if this
1473 element is present in an XRD advertising SAML authority resolution service (or any other
1474 service), and the client has reason to trust this XRD, the client MAY use the associated
1475 ProviderID to correlate the contents of this element with a signed response.

1476 To assist resolvers in using this key discovery mechanism, it is important that trusted authority
1477 resolution services be configured to sign responses in such a way that the signature can be
1478 verified using the correlated `ds:KeyInfo` element. For more information, see [SAML].

1479 9.3 HTTPS+SAML

1480 9.3.1 Service Type and Service Media Type

1481 The protocol defined in this section is identified by the values in Table 17.

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

1482 *Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.*

1483 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST
1484 use the Service Type identifier and Service Media Type identifier defined in Table 17 (including
1485 the `https=true` and `saml=true` parameters). In addition, the identifier authority MUST use an
1486 HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1487 9.3.2 Protocol

1488 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- 1489 1. All authority resolution service endpoints MUST be selected using the values defined in
1490 Table 17.
- 1491 2. All authority resolution requests and responses, including the starting request to a
1492 community root authority, MUST conform to both the requirements of the HTTPS trusted
1493 resolution protocol defined in section 9.1 and the SAML trusted resolution protocol
1494 defined in section 9.2.
- 1495 3. All authority resolution requests MUST contain an HTTPS Accept header with the media
1496 type identifier defined in Table 17 (including both the `https=true` and `saml=true`
1497 parameters). This MUST be interpreted as the value of the Resolution Output Format
1498 input parameter.
- 1499 4. If the resolver finds that an authority in the resolution chain does not support both HTTPS
1500 and SAML, the resolver MUST return a 23x error as defined in section 14.

1501 10 Proxy Resolution

1502 The preceding sections have defined XRI resolution as a set of logical functions that may
1503 implemented via a local resolver interface. This section defines a mapping of these functions to
1504 an HTTP(S) interface for remote invocation. This mapping is based on a standard syntax for
1505 expressing an XRI as an HTTP URI, called an *HXRI*, as defined in section 10.2. This includes a
1506 method of passing the other XRI resolution input parameters as query parameters in the HXRI.

1507 Proxy resolution is useful for many reasons:

- 1508 • Offloading XRI resolution and service endpoint selection processing from a client to an
1509 HTTP(S) server.
- 1510 • Optimizing XRD caching for a resolution community (a *caching proxy resolver*). Proxy
1511 resolvers SHOULD use caching to resolve the same QXRI or QXRI components for multiple
1512 clients as defined in section 15.4.
- 1513 • Returning HTTP(S) redirects to clients such as browsers that have no native understanding
1514 of XRIs but can process HXRIs. This provides backwards compatability with the large
1515 installed base of existing HTTP clients.

1516 10.1 Service Type and Media Types

1517 The protocol defined in this section is identified by the values in Table 18.

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

1518 Table 18: Service Type and Service Media Type values for proxy resolution.

1519 A proxy resolution service endpoint advertised in an XRDS document MUST use the Service
1520 Type identifier and Service Media Type identifiers defined in Table 18 (including the optional
1521 media type parameters). In addition, if the media type parameter `https=true` is included, the
1522 identifier authority MUST offer at least one HTTPS URI as the value of the `xrd:URI` element(s)
1523 for this service endpoint.

1524 It may appear to be of limited value to advertise proxy resolution service in an XRDS document if
1525 a resolver must already know how to perform local XRI resolution in order to retrieve this
1526 document. However advertising a proxy resolution service in the XRDS document for a
1527 community root authority (sections 8.1.3 and 8.1.4) can be very useful for applications that need
1528 to consume XRI proxy resolution services or automatically generate HXRIs for resolution by non-
1529 XRI-aware clients in that community. Those applications may discover the current URI(s) and
1530 media type capabilities of a proxy resolver from this source.

1531 10.2 HXRIs

1532 The first step in an HTTP binding of the XRI resolution interface is to specify how the QXRI
1533 parameter is passed within an HTTP(S) URI. Besides providing a binding for proxy resolution,
1534 defining a standard syntax for expressing an XRI as an HTTP XRI (HXRI) has two other benefits:

- 1535 • It allows XRIs to be used anyplace an HTTP URI can appear, including in Web pages,
1536 electronic documents, email messages, instant messages, etc.
- 1537 • It allows XRI-aware processors and search agents to recognize an HXRI and extract the
1538 embedded XRI for direct resolution, processing, and indexing.

Comment [DSR16]: In the future
this section will move to XRI Syntax
3.0.

1539 To make this syntax as simple as possible for XRI-aware processors or search agents to
1540 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority segment that
1541 begins with the domain name segment "xri.". The QXRI is then appended as the entire local
1542 path (and query component, if present). The QXRI MUST NOT include the "xri://" prefix and
1543 MUST be in URI-normal form as defined in **[XRISyntax]**. (If a proxy resolver receives an HXRI
1544 containing a QXRI beginning with an "xri://" prefix, it SHOULD follow Postel's Law¹ by
1545 removing it before continuing.) In essence, the proxy resolver URI (including the forward slash
1546 after the domain name) serves as a machine-readable prefix for an absolute XRI in URI-normal
1547 form.

1548 The normative ABNF for an HXRI is defined below based on the *ireg-name*, *xri-hier-part*,
1549 and *iquery* productions defined in **[XRISyntax]**. Authors whose XRIs need to be understood by
1550 non-XRI-aware clients SHOULD publish them as HTTP URIs conforming to this HXRI production.

```
1551 HXRI           = proxy-resolver "/" QXRI
1552 proxy-resolver = ( "http://" / "https://" ) proxy-reg-name
1553 proxy-reg-name = "xri." ireg-name
1554 QXRI          = xri-hier-part [ "?" i-query ]
```

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

1555 URI processors that recognize XRIs SHOULD interpret the local part of an HTTP or HTTPS URI
 1556 (the path segments and optional query segment) that conforms to this ABNF as an XRI provided
 1557 the first segment of the path conforms to the xri-authority or iauthority productions in
 1558 [XRISyntax].

1559 For references to communities that offer public XRI proxy resolution services, see the XRI
 1560 Technical Committee home page at <http://www.oasis-open.org/committees/xri>.

Comment [DSR17]: OPEN ISSUE: pending.

1561 10.3 HXRI Query Parameters

1562 There are a total of five logical input parameters to XRI authority resolution and service endpoint
 1563 selection as defined in section 7.1:

- 1564 1. QXRI – the entire original XRI for which resolution is requested.
- 1565 2. Path String – the path component of the QXRI.
- 1566 3. Resolution Output Format – the requested media type of the resolution response (see
 1567 section 7.1.2).
- 1568 4. Service Type – the type of service requested for service endpoint selection (see section
 1569 7.1.3).
- 1570 5. Service Media Type – the type of media requested for service endpoint selection (see
 1571 section 7.1.4).

1572 In proxy resolution, these parameters are bound to an HTTP(S) interface using the conventional
 1573 web model of encoding them in an HTTP(S) URI, which in this case is an HXRI. The binding of
 1574 the logical parameter names to HXRI component parts is defined in Table 19.

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

1575 *Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.*

1576 Following are the rules for the use of the parameters specified in Table 19.

- 1577 1. The QXRI MUST be normalized as specified in section 10.2.
- 1578 2. If the original QXRI has an existing query component, the HXRI query parameters MUST
 1579 be appended to that query component. (Note that the query parameter names in Table
 1580 19 were chosen to minimize the probability of collision with any other existing query
 1581 parameter names.) After proxy resolution, the HXRI query parameters MUST
 1582 subsequently be removed from the QXRI query component. The existing QXRI query
 1583 component MUST NOT be altered in any other way, i.e., it must be passed through with
 1584 no changes in parameter order, escape encoding, etc.
- 1585 3. If the original QXRI does not have a query component, one MUST be added to pass any
 1586 HXRI query parameters. After proxy resolution, this query component MUST be entirely
 1587 removed.

- 1588 4. If the original QXRI had a null query component (only a leading question mark), or a
1589 query component consisting of only question marks, *one additional leading question mark*
1590 MUST be added before adding any HXRI query parameters. After proxy resolution, any
1591 HXRI query parameters and exactly one leading question mark MUST be removed. See
1592 the URI construction step defined in section 11.6.
- 1593 5. Each HXRI query parameter MUST be delimited from other parameters by an ampersand
1594 (“&”). Any occurrences of the character “&” within an input parameter (specifically the
1595 Service Type value) MUST be percent encoded prior to input and decoded upon output.
- 1596 6. Each HXRI query parameter MUST be delimited from its value by an equals sign (“=”).
- 1597 7. In an HXRI query parameter, the character + and the percent-encoded sequence %2B
1598 MUST be interpreted as the same character, therefore spaces in an HXRI query
1599 parameter MUST NOT be encoded as + signs.
- 1600 8. Any XRIs or IRIs used as values of HXRI query parameters MUST be in URI-normal form
1601 as defined in [XRISyntax].
- 1602 9. If any HXRI query parameter name is included but its value is empty, the value of the
1603 parameter MUST be considered null.
- 1604 10. For proxy resolution, any input parameter supplied explicitly via an HXRI query parameter
1605 MUST take precedence over the same parameter provided via an HTTP(S) header, even
1606 if the value of the HXRI query parameter is explicitly null. See the following section.

1607 10.4 HTTP(S) Accept Headers

1608 In proxy resolution, one XRI resolution input parameter, the Service Media Type (section 7.1.4)
1609 MAY be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. The
1610 following rules apply to this input:

- 1611 1. As described in section 14.1 of [RFC2616], the Accept header content type MAY consist
1612 of multiple media type identifiers. If so, the proxy resolver MUST choose only one to
1613 accept. A proxy resolver client SHOULD order media type identifiers according to the
1614 client’s preference and a proxy resolver server SHOULD choose the client’s highest
1615 preference.
- 1616 2. If the value of the Accept header content type is null, this MUST be interpreted as the
1617 value of the Service Media Type parameter.
- 1618 3. If the value of the Service Media Type parameter is explicitly set via the `_xrd_m` query
1619 parameter in the HXRI (including to a null value), this MUST take precedence over any
1620 value set via an HTTP(S) Accept header.

1621 10.5 Null Resolution Output Format

1622 Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a
1623 resolution request where the Resolution Output Format input parameter value is null—either
1624 because this parameter is absent or because it was explicitly set to null using the `_xrd_r` query
1625 parameter.

1626 If the value of the Resolution Output Format value is null, a resolver MUST act as if the following
1627 media type parameters had the following values: `https=false`, `saml=false`, `refs=true`,
1628 `sep=true`, and `nodefault=false`. In addition, the output MUST be an HTTP(S) redirect as
1629 defined in the following section.

1630 10.6 Outputs and HTTP(S) Redirects

1631 For all values of the Resolution Output Format parameter except null, a proxy resolver MUST
1632 follow the output rules defined in section 7.2.

1633 If the value of the Resolution Output Format is null, and the output is not an error, a proxy
1634 resolver MUST follow the rules for output of a URI List as defined in section 7.2.3. However
1635 instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as
1636 an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service
1637 Media Type parameter.

1638 If the output is an error, a proxy resolver SHOULD return a human-readable error message with a
1639 media type of either `text/plain` or `text/html`.

1640 This rule enables XRI proxy resolvers to serve clients that do not understand XRI syntax or
1641 resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the
1642 service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept
1643 header (if any) as described in section 10.4.

1644 **10.7 Differences Between Proxy Resolution Servers**

1645 An XRI proxy resolution request MAY be sent to any proxy resolver that will accept it. All XRI
1646 proxy resolvers SHOULD deliver uniform responses given the same QXRI and other input
1647 parameters. However because proxy resolvers may potentially need to make decisions about
1648 network errors, reference processing, and trust policies on behalf of the client they are proxying,
1649 and these decisions may be based on local policy, in some cases different proxy resolvers may
1650 return different results.

1651 **10.8 Combining Authority and Proxy Resolution Servers**

1652 The majority of DNS nameservers are recursing nameservers that answer both queries for which
1653 they are authoritative and queries which they must forward to other nameservers. The same
1654 applies to XRI architecture: in many cases the optimum configuration will be to combine an
1655 authority resolution service and a proxy resolution service in the same server. This server can
1656 publish a self-describing XRDS document (section 8.1.4) that advertises both its authority
1657 resolution and proxy resolution service endpoints. It can also optimize caching of XRDs for clients
1658 in its resolution community (see section 15.4).

1659 **11 Service Endpoint Selection**

1660 If the authority resolution phase is successful, the output is an XRDS document containing a final
1661 XRD describing the target authority. If requested, a resolver may perform an optional second
1662 phase of XRI resolution by processing this XRDS document to select a requested service
1663 endpoint. This section specifies the rules for this process.

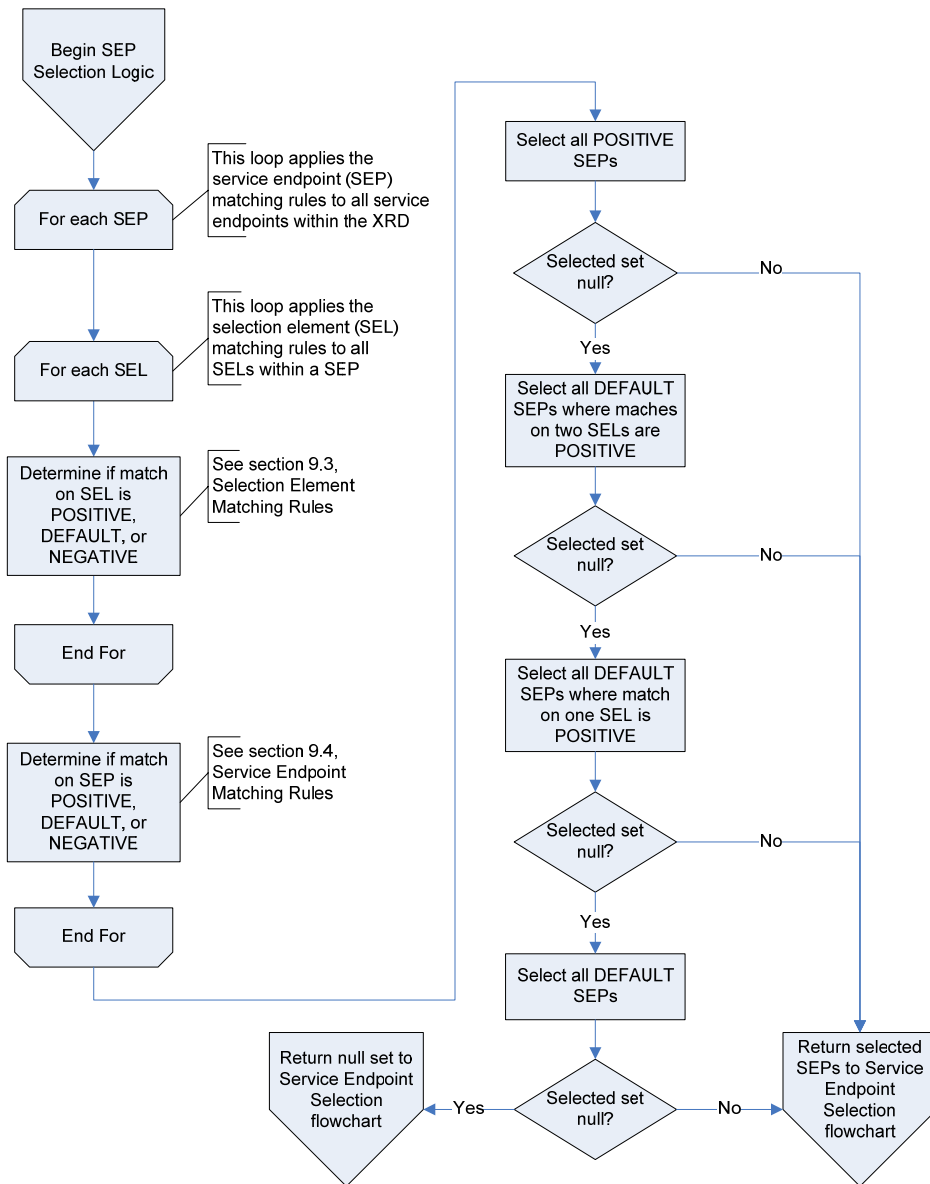
1664 **11.1 Processing Rules**

1665 Figure 7 (non-normative) shows the overall logical flow of the service endpoint selection process.

1668 Following are the normative rules for the overall service endpoint selection process:
1669 1. The inputs for service endpoint selection are defined in Table 8.
1670 2. For each `xrd:Service` element in the current XRD, selection MUST follow the service
1671 endpoint selection logic defined in section 11.2. The outcome of this process will be a
1672 selected set of zero or more service endpoints.
1673 3. If after applying the service endpoint selection rules no service endpoint is selected, a
1674 resolver MUST perform authority reference processing as defined in section 12.1 if the
1675 `refs` media type parameter is set to `true` or null, or if the parameter is absent. If the
1676 `refs` media type parameter is set to `false` and the XRD contains at least one
1677 `xrd:XRD/xrd:Ref` element that could be followed, the resolver MUST return a
1678 successful response with a status code of 101 `REF_NOT_FOLLOWED`.
1679 4. If an XRD does not include the requested service endpoint and does not include any
1680 `xrd:XRD/xrd:Ref` elements, the resolver MUST return an error with a status code of
1681 241 `SEP_NOT_FOUND` regardless of the value of the `refs` media type parameter.
1682 5. If after applying the service endpoint selection rules at least one service endpoint is
1683 selected, but it does NOT contain an `xrd:XRD/xrd:Service/xrd:URI` element and
1684 DOES contain an `xrd:XRD/xrd:Service/xrd:Ref` element, a resolver MUST
1685 perform service reference processing as defined in section 12.2.
1686 6. The output of service endpoint selection MUST be a valid Resolution Output Format as
1687 defined in Table 10. If this output is an XRDS Document, an XRD Document, or a URI
1688 List, it MUST conform to the output requirements defined in section 7.2. If the Resolution
1689 Output Format value is null and the output is an HTTP(S) redirect, the resolver MUST
1690 proceed as defined in section 10.6.

1691 11.2 Service Endpoint Selection Logic

1692 Selection of service endpoints (SEPs) within an XRD is managed using service endpoint
1693 selection elements (SEs). As defined in section 4.2.7, `xrd:XRD/xrd:Service` elements may
1694 contain three categories of selection elements: `xrd:Type`, `xrd:Path`, and `xrd:MediaType`.
1695 Figure 7 (non-normative) shows the overall flow of the service endpoint selection logic.



1696

1697 *Figure 8: Service endpoint (SEP) selection logic flowchart.*

1698 The following sections provide the normative rules for each section of this flowchart.

1699 **11.3 Selection Element Matching Rules**

1700 The first set of rules govern the matching of selection elements.

1701 **11.3.1 Selection Element Match Options**

1702 Within each service endpoint, there is a match option for each of the three categories of selection
 1703 elements. The three match options are defined in Table 20:

Match Option	Match Condition	Precedence
POSITIVE	A successful match based on the value of the <code>xrd:match</code> attribute as defined in 11.3.2 OR a successful match based the contents of the selection element as defined in sections 11.3.6 - 11.3.8.	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 11.3.3.	0
NEGATIVE	The selection element does not satisfy either condition above.	-1

1704 Table 20: Match options for selection elements.

1705 **11.3.2 The Match Attribute**

1706 All three service endpoint selection elements accept the optional `xrd:match` attribute. This
 1707 attribute gives XRDS authors precise control over selection of service endpoints based on the
 1708 QXRI and other resolution input parameters. An enumerated list of the values for this attribute is
 1709 defined in Table 21. If this attribute is present with one of these values, the contents of the
 1710 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_t</code> , <code>nodefault_p</code> or <code>nodefault_m</code> parameter is set to <code>true</code> for the applicable 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.

1711 Table 21: Enumerated values of the global match attribute and corresponding matching rules.

1712 *Backwards Compatability Note:* earlier working drafts of this specification included the values
 1713 `match="none"` and `match="contents"`. Both are deprecated. The former is no longer
 1714 supported and the latter is now the default behaviour of any selection element that does not
 1715 include the match attribute or the value of this attribute is empty.

1716 **11.3.3 Absent Selection Element Matching Rule**

1717 If a service endpoint does not contain at least one instance of a particular category of selection
 1718 element, it MUST be considered equivalent to the service endpoint having a DEFAULT match on
 1719 that category of selection element UNLESS the overridden by the `nodefault` parameter as specified
 1720 in Table 21.

1721 11.3.4 Empty Selection Element Matching Rule

1722 If a selection element is present in a service endpoint but the element is empty, and if the element
1723 does not contain an `xrd:match` attribute or the value of this attribute is empty, it MUST be
1724 considered equivalent to having a DEFAULT match on this selection element UNLESS the
1725 overridden by the `nodefault` parameter as specified in Table 21.

1726 11.3.5 Multiple Selection Element Matching Rule

1727 Each service endpoint has only one match option for each category of selection element.
1728 Therefore if a service endpoint contains more than one instance of the same category of selection
1729 element (i.e., more than one `xrd:Type`, `xrd:Path`, or `xrd:MediaType` element), matching
1730 MUST be based on the selection element with the highest precedence match option as defined in
1731 Table 20.

1732 11.3.6 Type Element Matching Rules

1733 The following rules apply to matching the value of the input Service Type parameter with the
1734 contents of a non-empty `xrd:XRD/xrd:Service/xrd:Type` element when its `xrd:match`
1735 attribute is absent or empty.

- 1736 1. The values of the Service Type parameter and the `xrd:XRD/xrd:Service/xrd:Type`
1737 element SHOULD be normalized according to the requirements of their identifier scheme
1738 prior to input. In particular, if an XRI, IRI, or URI does not include a local part (a path
1739 and/or query component), a trailing forward slash MUST be assumed after the authority
1740 component. In all other cases, a trailing forward slash MUST NOT be assumed, and
1741 therefore is significant in comparisons. In addition, if the value is an XRI or an IRI it
1742 MUST be in URI-normal form as defined in section 4.4. XRI resolvers MAY perform
1743 normalization of these values but MUST NOT be required to do so. As a best practice,
1744 service architects SHOULD assign identifiers for service types that are easy to match and
1745 do not require normalization.
- 1746 2. To result in a POSITIVE match, the values MUST be equivalent according to the
1747 equivalence rules of the applicable identifier scheme. Any other result is a NEGATIVE
1748 match.

1749 11.3.7 Path Element Matching Rules

1750 The following rules apply to matching the value of the input Path String with the contents of a
1751 non-empty `xrd:XRD/xrd:Service/xrd:Path` element when its `xrd:match` attribute is
1752 absent or empty.

- 1753 1. If the value is a relative XRI or an IRI it MUST be in URI-normal form as defined in
1754 section 4.4.
- 1755 2. The QXRI Path String being matched MUST NOT include the forward slash separating
1756 an XRI authority segment from the path. Any subsequent forward slash, including trailing
1757 forward slashes, MUST be significant in comparisons.
- 1758 3. Equivalence comparison MUST be performed using Caseless Matching as defined in
1759 section 3.13 of **[Unicode]**, with the exception that it is not necessary to perform
1760 normalization after case folding.
- 1761 4. To result in a POSITIVE match, the value of the Path String MUST be a *subsegment*
1762 *stem match* with the contents of the Path element. Any other result MUST be a
1763 NEGATIVE match. A subsegment stem match is defined as the entire Path String being
1764 character-for-character equivalent with any continuous sequence of subsegments or
1765 segments as defined in **[XRISyntax]** (including empty subsegments and empty
1766 segments) of the contents of the Path element beginning from the most significant
1767 (leftmost) subsegment.

1768 Examples of the this rule are shown in table [TODO] below.

1769 11.3.8 MediaType Element Matching Rules

1770 The following rules apply to matching the value of the input Service Media Type parameter with
1771 the contents of of a non-empty `xrd:XRD/xrd:Service/xrd:MediaType` element when its
1772 `xrd:match` attribute is absent or empty.

- 1773 1. The values of the Service Media Type parameter and the `xrd:MediaType` element
1774 SHOULD be normalized according to the rules for media types in section 3.7 of
1775 [RFC2616] prior to input. (The rules are that type and subtype names are case-
1776 insensitive, but parameter values may or may not be case-sensitive depending on the
1777 semantics of the parameter name. XRI Resolution Output Format parameters are case-
1778 insensitive.) XRI resolvers MAY perform normalization of these values but MUST NOT be
1779 required to do so.
- 1780 2. To be a POSITIVE match, the values MUST be character-for-character equivalent. Any
1781 other result is a NEGATIVE match.

1782 11.4 Service Endpoint Matching Rules

1783 The next set of matching rules govern the matching of service endpoints based on the matches of
1784 the selection elements they contain.

1785 11.4.1 Service Endpoint Match Options

1786 For each service endpoint in an XRD, there are three match options as defined in Table 22:

Match Option	Condition
POSITIVE	Meets the Select Attribute Match Rule (section 11.4.2) or the All Positive Match Rule (section 11.4.3).
DEFAULT	Meets the Default Match Rule (section 11.4.4).
NEGATIVE	The service endpoint does not satisfy either condition above.

1787 *Table 22: Match options for service endpoints.*

1788 11.4.2 Select Attribute Match Rule

1789 All three service endpoint selection elements accept the optional `xrd:select` attribute. This
1790 attribute is a Boolean value used to govern selection of the containing service endpoint according
1791 to the following rule. If service endpoint contains a selection element with a POSITIVE match as
1792 defined in section 11.3, and the value of this selection element's `xrd:select` attribute is `true`,
1793 the service endpoint automatically MUST be a POSITIVE match, i.e., all other selection elements
1794 for this service endpoint MUST be ignored.

1795 11.4.3 All Positive Match Rule

1796 If a service endpoint has a POSITIVE match on all three categories of selection elements
1797 (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) as defined in section 11.3, the service endpoint
1798 MUST be a POSITIVE match. If even one of the three selection element match types is not
1799 POSITIVE, this rule fails.

1800 11.4.4 Default Match Rule

1801 If a service endpoint fails the All Positive Match Rule, but none of the three categories of selection
1802 elements has a NEGATIVE match as defined in section 11.3, the service endpoint MUST be a
1803 DEFAULT match.

1804 11.5 Service Endpoint Selection Rules

1805 The final set of rules governs the selection of service endpoints based on their matches.

1806 11.5.1 Positive Match Rule

1807 After applying the matching rules to service endpoints in section 11.4, all service endpoints that
1808 have a POSITIVE match MUST be selected. Only if there are no service endpoints with a
1809 POSITIVE match is the Default Match Rule invoked.

1810 11.5.2 Default Match Rule

1811 If the Positive Match Rule above fails, then the service endpoints with a DEFAULT match that
1812 have the highest number of POSITIVE matches on each category of selection element MUST be
1813 selected. This means:

- 1814 1. The service endpoints in the DEFAULT set that have two POSITIVE selection element
1815 matches MUST be selected.
- 1816 2. If the previous set is empty, the service endpoints in the DEFAULT set that have one
1817 POSITIVE selection element match MUST be selected.
- 1818 3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.
- 1819 4. If the previous set is empty, no service endpoint is selected and the return set is null.

1820 11.6 Pseudocode

1821 The following pseudocode provides a precise description of the service endpoint selection logic.
1822 The pseudocode is normative, however if there is a conflict between it and the rules stated in the
1823 preceding sections, the preceding sections shall prevail.

1824 The pseudocode uses nine Boolean flags to record the match state for each category of selection
1825 element (SEL) in a service endpoint (SEP):

- 1826 • Postive.Type
- 1827 • Postive.Path
- 1828 • Positive.Mediatype
- 1829 • Default.Type
- 1830 • Default.Path
- 1831 • Default.Mediatype
- 1832 • Matched.Type
- 1833 • Matched.Path
- 1834 • Matched.Mediatype

1835 Note that the complete set of nine SEL match flags is needed for each SEP. The pseudocode first
1836 does a loop through all SEPs in the XRD to:

- 1837 1. Set the SEL match flags according to the rules specified in section 11.3;
- 1838 2. Process the SEL match flags to apply the SEP matching rules specified in section 11.4;
- 1839 3. Apply the positive SEP selection rule specified in section 11.5.1.

Formatted: Normal, Space After: 0 pt

Formatted: List Number, Space After: 0 pt, Numbered + Level: 1 + Numbering Style: 1, 2, 3, ... + Start at: 1 + Alignment: Left + Aligned at: 18 pt + Tab after: 36 pt + Indent at: 36 pt, Don't adjust space between Latin and Asian text

1840 [After this loop is complete, the pseudocode tests to see if default SEP selection processing is](#)
1841 [required. If so, it performs a second loop applying the default SEP selection rules specified in](#)
1842 [section 11.5.2.](#)

← - - - - **Formatted:** Normal, Space After: 0
pt, Don't adjust space between Latin
and Asian text

1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900

```
FOR EACH SEP
  CREATE set of SEL match flags
  SET all flags to FALSE
  FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
    SET Matched.x=TRUE
    IF match on this SEL is POSITIVE
      IF select="true" ;see 11.4.2
        ADD SEP TO SELECTED SET
      NEXT SEP
    ELSE
      SET Positive.x=TRUE
      NEXT SEL
    ENDIF
    ELSEIF match on this SEL is DEFAULT ;see 10.3.2 & 11.3.4
      IF Positive.x=TRUE ;see 11.3.5
        NEXT SEL
      ELSEIF nodefault="x" ;see 10.3.2
        NEXT SEL
      ELSE
        SET Default.x=TRUE
        NEXT SEL
      ENDIF
    ELSEIF match on this SEL is NEGATIVE ;see 11.3.1
      NEXT SEL
    ENDIF
  ENDFOR
  IF Matched.x=FALSE ;see 11.3.3
    IF nodefault_x != TRUE ;see 10.3.2
      SET Default.x=TRUE
    ENDIF
  ENDIF
  IF Positive.Type=TRUE AND
    Positive.Path=TRUE AND
    Positive.Mediatype=TRUE ;see 11.4.3
    ADD SEP TO SELECTED SET
  NEXT SEP
  ELSEIF SELECTED SET != EMPTY ;see 11.5.1
    NEXT SEP
  ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
    (Positive.Path=TRUE OR Default.Path=TRUE) AND
    (Positive.MediaType=TRUE OR Default.MediaType=TRUE)
    ADD SEP TO DEFAULT SET ;see 11.4.4
  ENDIF
ENDFOR
IF SELECTED SET != EMPTY ;see 11.5.1
  RETURN SELECTED SET
ENDIF
IF DEFAULT SET != EMPTY ;see 11.5.2
  FOR EACH SEP IN DEFAULT SET
    IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
      (Positive.Type=TRUE AND Positive.MediaType=TRUE) OR
      (Positive.Path=TRUE AND Positive.MediaType=TRUE)
      ADD SEP TO SELECTED SET
    ENDIF
  ENDFOR
IF SELECTED SET != EMPTY
  RETURN SELECTED SET
```

1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915

```

ENDIF
FOR EACH SEP IN DEFAULT SET
  IF Positive.Type=TRUE OR
  Positive.Path=TRUE OR
  Positive.MediaType=TRUE
    ADD SEP TO SELECTED SET
  ENDFOR
ENDIF
IF SELECTED SET != EMPTY
  RETURN SELECTED SET
ELSE
  RETURN DEFAULT SET
ENDIF
ENDIF
RETURN EMPTY SET

```

Comment [DSR18]: This section rewritten in ED04 for improved clarity per suggestion from Victor. No normative changes.

1916 **11.7 Construction of Service Endpoint URIs**

1917 If the output is a URI List, the final step in the service endpoint selection process is construction
1918 of the service endpoint URI(s). This is governed by the `xrd:append` attribute of each `xrd:URI`
1919 element. The values of this attribute are shown in Table 23.

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>of the QXRI, defined as being</i> one of three cases: a) If only a path is present, the path string <i>plus the leading forward slash</i> b) If only a query is present, the query string <i>plus the leading question mark</i> c) If both a path and a query are present, the entire combination of the path string <i>plus the leading forward slash</i> and the query string <i>plus the leading question mark</i>
authority	Authority string only (including the community root subsegment) <i>without the trailing forward slash</i>
path	Path string <i>plus the leading forward slash</i>
query	Query string <i>plus the leading question mark</i>
qxri	Entire QXRI

1920 *Table 23: Values of the append attribute and the corresponding QXRI component to append.*

1921 If the `xrd:append` attribute is absent or its value is empty, the default value is `none`. Following
1922 are the rules for construction of the final service endpoint URI based on the value of the
1923 `xrd:append` attribute. *Note that these rules must be followed closely in order to give XRD*
1924 *authors precise control over construction of service endpoint URIs.*

- 1925 1. If the value is `none`, the exact contents of the `xrd:URI` element **MUST** be returned
1926 directly without any further processing.
- 1927 2. For any other value, the exact value of the QXRI component specified in Table 23,
1928 *including any leading delimiter(s) and without any additional escaping or percent*
1929 *encoding* **MUST** be appended directly to the exact contents of the `xrd:URI` element
1930 *including any trailing delimiter(s)*. If the value of the QXRI component specified in Table
1931 23 consists of only a leading delimiter, then this value **MUST** be appended according to

1932 these rules. If the value of the QXRI component specified in Table 23 is null, then the
1933 contents of the `xrd:URI` element MUST be returned directly exactly as if the value of the
1934 `xrd:append` attribute was none.

1935 3. If any query parameters for XRI proxy resolution were added to an existing QXRI query
1936 component as defined in section 10.3, these query parameters MUST be removed prior
1937 to performing the append operation as also defined in section 10.3. In particular, if after
1938 removal of these query parameters the QXRI query component consists of only a *string*
1939 *of one or more question marks* (the delimiting question mark plus zero or more additional
1940 question marks) then *exactly one question mark* MUST also be removed. This preserves
1941 the query component of the original QXRI if it was null or contained only question marks.

1942 **IMPORTANT:** Construction of HTTP(S) URIs for XRI authority resolution service endpoints is
1943 defined in section 8.1.8. Note that this involves an additional step taken after all URI construction
1944 processing in this section is complete. In other words, if the URI element of an XRI authority
1945 resolution service endpoint includes an `append` attribute, the Next Authority Service URI should
1946 be fully constructed according to the algorithm in this section before appending the Next Authority
1947 String as defined in section 8.1.8.

1948

12 Reference Processing

1949

As defined in section 5.2.5, the `xrd:Ref` element plays a special role in XRDS architecture. Its purpose is to enable parent authorities to assert that another XRD exists that describes the target resource, and which MAY contain different metadata than the current XRD. XRDS Ref architecture explicitly supports distributed, polyarchical description of resources by multiple cooperating authorities.

1951

1952

1953

1954

A `xrd:Ref` element MAY be used at the authority level of an XRD (as a child of the root `xrd:XRD` element) or the service level (as a child of the root `xrd:XRD/xrd:Service` element).

1955

1956

At the authority level, an *authority reference* is an assertion that another parent authority is authorized to describe the child authority with an XRDS document, and that a resolver MUST consider the XRDs from both authorities as a "logical union".

Formatted: Font: Italic

1957

1958

At the service level, a *service reference* is an assertion that another XRDS document describes the target authority in the context of a specific service, and that a resolver MUST consider service elements satisfying the same service endpoint selection criteria in both XRDs as a "logical union".

Formatted: Font: Italic

1959

1960

1961

1962

12.1 Authority References

1963

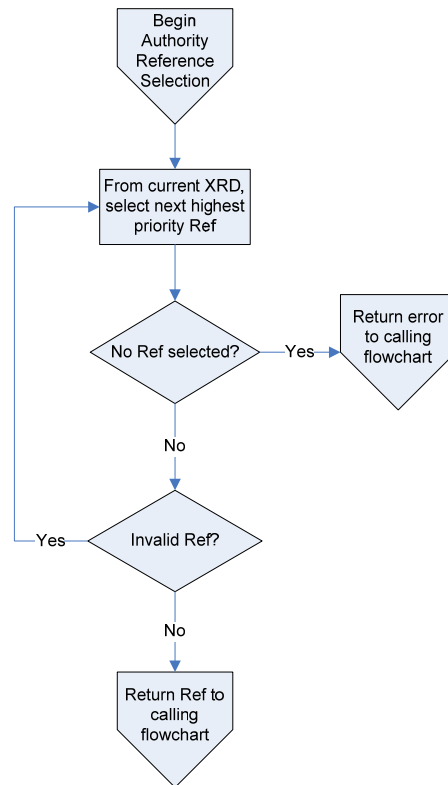
This section contains the normative rules for processing of authority references in XRI resolution.

1964

12.1.1 Processing Rules

1965

Figure 9 is an overview of the logical flow of selecting an authority reference for processing.



1966

1967 *Figure 9: Flowchart for selecting an authority reference for processing.*

1968 Following are the normative rules that apply to authority reference processing:

- 1969 1. Authority reference processing will only be performed if the `refs` media type parameter
 1970 (Table 6) is omitted or its value is `true`. If the value is `false` and the XRD contains at
 1971 least one `xrd:Ref` element that could be followed, the resolver MUST return a response
 1972 with a status code of 101 `REF_NOT_FOLLOWED`.
- 1973 2. The resolver MUST begin by selecting the highest priority `xrd:XRD/xrd:Ref` element in
 1974 the current XRD. If not found, the result is an error and the resolver MUST proceed as
 1975 defined in section 8 for authority resolution or section 11 for service endpoint selection.
- 1976 3. The contents of the selected `xrd:XRD/xrd:Ref` element MUST be a valid HTTP(S) URI
 1977 or a valid QXRI as defined in section 7.1.1. If not, it MUST be ignored and step 1
 1978 repeated to select the next highest priority reference.
- 1979 4. If the value is an HTTP(S) URI and the original XRI resolution parameter specified that
 1980 `https=true`, then the value MUST be an HTTPS URI. If not, it is an invalid reference and
 1981 MUST be ignored and step 1 repeated to select the next highest priority reference.
- 1982 5. If the value of the selected `xrd:XRD/xrd:Ref` element is an HTTP(S) URI, the resolver
 1983 MUST resolve it to a new XRDS document as specified in section 5.
- 1984 6. If the value of the selected `xrd:XRD/xrd:Ref` element is a QXRI, the resolver MUST
 1985 begin resolution of a new XRDS document beginning with the community root authority of
 1986 the authority reference XRI as defined in section 8.1.3. The resolver MUST use the same
 1987 resolution input parameters as for the original QXRI. For reference processing to
 1988 complete successfully, the resolver MUST complete resolution of the entire Authority

- 1989 String of the reference XRI (including following any further authority references if
 1990 necessary). If the reference XRI includes a Path String or Query String (including any
 1991 resolution query parameters), they MUST be ignored.
- 1992 7. If authority reference processing is successful and the Resolution Output Format is an
 1993 XRDS document, the XRDS document resulting from authority reference processing
 1994 MUST be nested inside the parent XRDS document as defined in section 12.1.2.
 - 1995 8. If authority reference processing is successful and the Resolution Output Format is an
 1996 XRD document, the output MUST be the final XRD document returned.
 - 1997 9. If authority reference processing is successful and the Resolution Output Format is a URI
 1998 List or an HTTP(S) redirect, it MUST be based on the final XRD document returned.
 - 1999 10. If authority reference processing fails, the resolver's work is complete and it MUST return
 2000 an error as defined in section 14. Since authority reference processing is another iteration
 2001 of authority resolution, it will typically be an authority resolution error.

2002 12.1.2 Nesting XRDS Documents

2003 If authority reference processing is successful, it will produce a new XRDS document that
 2004 describes the reference. If the final requested Resolution Output format is NOT an XRDS
 2005 document, this XRDS document is only needed to obtain the metadata necessary to continue
 2006 resolution. However, if the final requested Resolution Output Format is an XRDS document, this
 2007 new XRDS document MUST be included in the containing XRDS document immediately following
 2008 the `xrd:XRD` element that contains the `xrd:Ref` element being followed. In addition, the
 2009 `xrds:XRDS/@xrds:ref` attribute of this nested XRDS document MUST be set to the exact
 2010 value of the `xrd:XRD/xrd:Ref` element it describes.

2011 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
 2012 original QXRI even if resolution traverses authority references. Note that nested XRDS
 2013 documents do not include an XRD for the community root subsegment because this is part of the
 2014 configuration of the resolver.

2015 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD with an
 2016 `xml:id` attribute value matching the `xml:id` attribute value of any previous XRD in the chain of
 2017 resolution requests beginning with the original QXRI, the resolver MUST replace this XRD with an
 2018 empty XRD element. The resolver MUST set this empty element's `xrd:idref` attribute value to
 2019 the value of the `xrd:XRD/xml:id` attribute of the matched XRD element. This prevents
 2020 conflicting `xrd:XRD/xml:id` values.

2021 In the following example the original query XRI is `xri://@a*b*c`. The XRD for `xri://@a*b`
 2022 does not contain an authority resolution service endpoint but includes an authority reference to
 2023 `xri://@x*y`. The elements and attributes specific to authority reference processing are shown
 2024 in bold.

```

2025 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2026   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2027     <Query>*a</Query>
2028     ...
2029     <Service>
2030       <Type>xri://$res*auth*($v*2.0)</Type>
2031       <URI>http://a.example.com</URI>
2032     </Service>
2033   </XRD>
2034   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2035     <Query>*b</Query>
2036     ...
2037     <Ref>xri://@x*y</Ref>
2038     <Service>
2039       ...no authority resolution service endpoint...
2040     </Service>
  
```

```

2041 </XRD>
2042 <XRDS ref="xri://@x*y">
2043   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2044     <Query>*x</Query>
2045     ...
2046     <Service>
2047       <Type>xri://$res*auth*($v*2.0)</Type>
2048       <URI>http://x.example.com</URI>
2049     </Service>
2050   </XRD>
2051   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2052     <Query>*y</Query>
2053     ...
2054     <Service>
2055       <Type>xri://$res*auth*($v*2.0)</Type>
2056       <URI>http://y.example.com</URI>
2057     </Service>
2058   </XRD>
2059 </XRDS>
2060 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2061   <Query>*c</Query>
2062   ...
2063   <Service>
2064     ...final service endpoints described here...
2065   </Service>
2066 </XRD>
2067 </XRDS>

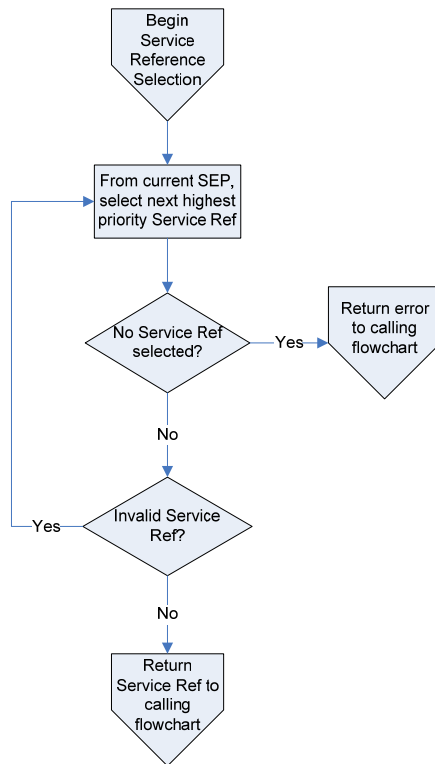
```

2068 12.2 Service References

2069 This section contains the normative rules for processing of service references in XRI resolution.

2070 12.2.1 Processing Rules

2071 Figure 9 is an overview of the logical flow of selecting a service reference for processing. The
 2072 process is essentially identical to that of Figure 9 for authority reference processing.



2073

2074 *Figure 10: Flowchart for selecting a service reference for processing.*

2075 Following are the normative rules that apply to service reference processing:

- 2076 1. If service endpoint selection is requested and the highest priority service endpoint in the
 2077 final XRD DOES NOT contain an `xrd:XRD/xrd:Service/xrd:URI` element but
 2078 DOES contain at least one `xrd:XRD/xrd:Service/xrd:Ref` element, the resolver
 2079 MUST begin reference processing by selecting the highest priority
 2080 `xrd:XRD/xrd:Service/xrd:Ref` element in the XRD.
- 2081 2. The contents of the selected `xrd:XRD/xrd:Service/xrd:Ref` element MUST be a
 2082 valid HTTP(S) URI. If not, it MUST be ignored and step 1 repeated to select the next
 2083 highest priority reference.
- 2084 3. If the original XRI resolution parameters specified that `https=true`, then the contents of
 2085 the selected `xrd:XRD/xrd:Service/xrd:Ref` element MUST be a valid HTTPS URI.
 2086 If not, it is an invalid service reference and MUST be ignored and step 1 repeated to
 2087 select the next highest priority reference.
- 2088 4. Once a valid service reference has been selected, if the
 2089 `xrd:XRD/xrd:Service/xrd:Ref` element includes the `xrd:append` attribute, the
 2090 resolver MUST construct the final HTTP(S) URI as defined in section 11.7.
- 2091 5. The resolver MUST directly request a new XRDS document from the final HTTP(S) URI
 2092 as defined in section 8.1.2. For service processing to complete successfully, the resolver
 2093 MUST also complete service endpoint selection (including following any further service
 2094 references if necessary) using the same service endpoint selection parameters as the
 2095 original query.

- 2096 6. If service reference processing is successful and the Resolution Output Format is an
 2097 XRDS document, the XRD document resulting from service reference processing **MUST**
 2098 sequentially follow the XRD containing the service reference as defined in section 12.2.2
- 2099 7. If service reference processing is successful and the Resolution Output Format is an
 2100 XRD document, the output **MUST** be the final XRD document returned.
- 2101 8. If service reference processing is successful and the Resolution Output Format is a URI
 2102 List or an HTTP(S) redirect, it **MUST** be based on the final XRD document returned.

2103 12.2.2 Adding XRD Documents

2104 If a service reference is followed successfully, it will produce an XRDS document containing an
 2105 XRD element with a service endpoint satisfying the original service endpoint selection
 2106 parameters. This XRD element **MUST** be included in the containing XRDS document immediately
 2107 following the `xrd:XRD` element containing the service reference being followed. In addition, the
 2108 resolver **MUST** set the value of the `xrd:XRD/@xrd:ref` attribute of this XRD document to the
 2109 value of the `xrd:XRD/xrd:Service/xrd:Ref` element used to request it.

2110 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
 2111 original QXRI even if resolution traverses one or more service references.

2112 For SAML trusted resolution, two special rules apply to the XRD elements resulting from service
 2113 reference processing:

- 2114 1. If an XRD is returned with an `xml:id` attribute value matching the `xml:id` attribute
 2115 value of any previous XRD in the chain of resolution requests beginning with the original
 2116 QXRI, the resolver **MUST** replace this XRD with an empty XRD element. The resolver
 2117 **MUST** set this empty element's `xrd:idref` attribute value to the value of the
 2118 `xrd:XRD/xml:id` attribute of the matched XRD element. This prevents conflicting
 2119 `xrd:XRD/xml:id` values.
- 2120 2. The `xrd:XRD/@xrd:ref` attribute appended to the XRD by the resolver **MUST NOT** be
 2121 included in the SAML signature validation processing defined in section 9.2.4.

2122 In the following example the original query XRI is `xri://a*b*c`. The XRD for `xri://a*b`
 2123 contains an authority resolution service endpoint without a URI element but with a service
 2124 reference to `http://x.example.com/xri/`. This produces a subsequent XRD with the
 2125 requested service endpoint type and URI element. The elements and attributes specific to service
 2126 reference processing are shown in bold.

```

2127 <XRDS xmlns="xri://$xrds" ref="xri://a*b*c">
2128   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2129     <Query>*a</Query>
2130     ...
2131     <Service>
2132       <Type>xri://$res*auth*($v*2.0)</Type>
2133       <URI>http://a.example.com</URI>
2134     </Service>
2135   </XRD>
2136   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2137     <Query>*b</Query>
2138     ...
2139     <Service>
2140       <Type>xri://$res*auth*($v*2.0)</Type>
2141       ...no URI element...
2142       <Ref append="qxri">http://x.example.com/xri/</Ref>
2143     </Service>
2144   </XRD>
2145   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0"
2146     ref="http://x.example.com/xri/@a*b*c">
2147     <Service>
  
```

```
2148         <Type>xri://$res*auth*($v*2.0)</Type>
2149         <URI>http://b.example.com</URI>
2150     </Service>
2151 </XRD>
2152 </XRDS>
2153 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2154     <Query>*c</Query>
2155     ...
2156     <Service>
2157         ...final service endpoints described here...
2158     </Service>
2159 </XRD>
2160 </XRDS>
```

2161

13 Synonym Verification

2162
2163
2164
2165

For security purposes, consuming applications often need be able to verify the binding between the query identifier (the identifier resolved to an XRDS document) and any synonyms for the query identifier asserted in the final XRD. This applies in particular to canonical synonyms because these are the identifiers consuming applications are recommended to persist.

2166
2167
2168
2169
2170
2171

As described in section 5, LocalIDs and Refs are self-verifying, since they are self-asserted by the parent authority in an XRD. Consuming applications may request XRI resolvers to perform automatic verification of CanonicalID and CanonicalEquivID synonyms as specified in section 13.1 below. Although XRI resolvers do not perform automatic verification of EquivID synonyms, this verification can easily be performed by consuming applications using one additional resolution call as specified in section 13.5.

2172

13.1 CanonicalID Verification

2173
2174
2175
2176
2177

While consuming applications may perform their verification of CanonicalID and CanonicalEquivID synonyms, 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.

2178
2179

1. If the value of the `xrd:XRD/xrd:CanonicalID` element is an HTTP(S) URI, it MUST be verified as specified in section 13.1.1.

2180
2181

2. If the value of the `xrd:XRD/xrd:CanonicalID` element is an XRI, it MUST be verified as specified in section 13.1.2.

2182
2183
2184

3. If the value of the `xrd:XRD/xrd:CanonicalID` element is any other identifier, CanonicalID verification fails and the resolver MUST return the CanonicalID verification failure status code specified in section 13.3.

2185
2186
2187
2188

4. If CanonicalID verification succeeds but the final XRD in the resolution chain also contains a `xrd:XRD/xrd:CanonicalEquivID` element, it MUST also be verified as specified in section 13.2. The resolver MUST return the status code specified in section 13.3.

2189
2190
2191

5. In all cases, since synonym verification depends on trusting each authority in the resolution chain, trusted resolution (section 9) SHOULD be used with either `https=true` and/or `saml=true` to provide additional assurance of the authenticity of the results.

2192
2193
2194
2195

6. In all cases, if service reference processing as defined in section 12.2 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, or else the resolver MUST return the status code specified in section 13.3.

2196
2197
2198
2199
2200
2201
2202
2203
2204
2205

IMPORTANT: There is no guarantee that all XRDs that describe the same child authority will return the same verified CanonicalID or CanonicalEquivID. Different parent authorities may assert different CanonicalIDs or CanonicalEquivIDs for the same child authority and all of these may all be verifiable. In addition, due to reference processing, the verified CanonicalID or CanonicalEquivID returned for an XRI MAY differ depending on the resolution input parameters. For example, as described in section 12.1.2, 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 or CanonicalEquivID for the child authority.

2206 13.1.1 HTTP(S) URI Verification Rules

2207 To verify that an HTTP(S) URI is a valid hierarchical CanonicalID synonym for a query identifier,
2208 an XRI resolver MUST verify that the following tests are successful:

- 2209 1. The query identifier MUST be an HTTP(S) URI.
 - 2210 • The query identifier MUST be resolved as specified in section 6.
- 2211 2. The asserted CanonicalID synonym MUST be an HTTP(S) URI equivalent to: a) the
2212 query identifier, or b) the query identifier plus a valid fragment as defined by [RFC3986].

2213 If the `xrd:XRD/xrd:CanonicalID` element contains any other HTTP(S) URI, or any other URI
2214 except an XRI, CanonicalID verification fails.

2215 13.1.2 XRI Verification Rules

2216 To verify that an XRI is a valid hierarchical CanonicalID synonym for a query identifier, an XRI
2217 resolver MUST verify that all the following tests are successful.

- 2218 1. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:ProviderID`
2219 element in the XRD from the community root authority MUST match the value of the
2220 `xrd:XRD/xrd:CanonicalID` element configure in the XRI resolver or available in a
2221 self-describing XRD from the community root authority (or its equivalent). See section
2222 8.1.4.
- 2223 2. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID`
2224 element MUST be a direct child the value of the `xrd:XRD/xrd:ProviderID` element.
- 2225 3. In all subsequent XRDs in the resolution chain, the value of the
2226 `xrd:XRD/xrd:CanonicalID` element MUST be a direct child of the value of the
2227 `xrd:XRD/xrd:CanonicalID` element in the parent XRD, i.e., it must only add a valid
2228 XRI subsegment to the parent CanonicalID value.
- 2229 4. If authority reference processing is required during resolution as specified in section 12.1,
2230 the three rules above MUST also apply to each nested XRDS document.

2231 In other words, the set of all possible XRIs that are valid CanonicalIDs synonyms for a query
2232 identifier forms a pure hierarchical identifier starting from the community root identifier as defined
2233 in section 5.1.1.

2234 13.2 CanonicalEquivID Verification

2235 CanonicalID verification also requires verification of a CanonicalEquivID but *only if it is present in*
2236 *the final XRD in the resolution chain*. Since CanonicalEquivID verification requires an extra
2237 resolution cycle, restricting automatic verification to the final XRD in the resolution chain ensures
2238 it will add at most one additional resolution cycle.

2239 To verify that either an HTTP(S) URI or an XRI is a valid hierarchical CanonicalEquivID synonym
2240 for a query identifier, an XRI resolver MUST verify that all the following tests are successful:

- 2241 1. CanonicalID verification as specified in section 13.1 MUST have completed successfully.
- 2242 2. The asserted CanonicalEquivID value MUST be an HTTP(S) URI or XRI and MUST
2243 successfully resolve per the rules in this specification.
- 2244 3. The final XRD in the resolution chain MUST contain a `xrd:XRD/xrd:EquivID` element
2245 whose value is equivalent to the value of the verified `xrd:XRD/xrd:CanonicalID`
2246 element in the XRD asserting the CanonicalEquivID synonym.

2247 See section 5.2.4 for recommendations regarding provisioning of an `xrd:XRD/xrd:EquivID`
2248 element in an XRD.

2249

13.3 Verification Error Codes and Recommended Identifiers

2250

When a resolution request includes the Resolution Output Format parameter `cid=true`, a resolver MUST NOT return `<Status code="100">SUCCESS</Status>` for any XRD in the resolution chain. Instead, for each XRD in the resolution chain, the resolver MUST return one of the status codes listed in Table 24.

2251

2252

2253

Code	Symbolic Error	Condition
110	CID_VERIFIED	A CanonicalID element is present and CanonicalID verification succeeded for this XRD
111	CID_VERIFICATION_FAILED	A CanonicalID element is present but CanonicalID verification failed for this XRD
112	CID_NOT_PRESENT	A CanonicalID element is not present in this XRD
113	CEID_VERIFIED	Both a CanonicalID and a CanonicalEquiVID element are present in this XRD and verification of both succeeded
114	CEID_VERIFICATION_FAILED	Both a CanonicalID and a CanonicalEquiVID element are present in this XRD and CanonicalID verification succeeded but CanonicalEquiVID verification failed

2254

Table 24: CanonicalID verification error codes.

2255

Based on these rules, a consuming application always knows from the status code on the final XRD in the resolution chain which synonym it SHOULD use to identify the target resource as specified in Table 25:

2256

2257

Code	Recommended Identifier	Reason
100	Query identifier	CanonicalID verification was not requested, so the query identifier is the only verified identifier
110	CanonicalID	A CanonicalID is present and verified but no CanonicalEquiVID is present, so the CanonicalID is recommended
111	Query identifier	CanonicalID verification failed, so the query identifier is the only verified identifier
112	Query identifier	A CanonicalID element is not present, so the query identifier is the only verified identifier
113	CanonicalEquiVID	Both a CanonicalID and a CanonicalEquiVID element are present and verified; the latter is recommended
114	CanonicalID	Both a CanonicalID and a CanonicalEquiVID element are present and CanonicalID verification succeeded but CanonicalEquiVID verification failed, so the CanonicalID is recommended

2258

Table 25: Recommended identifiers to use based on XRD status codes.

2259 13.4 Examples

2260 13.4.1 CanonicalID Verification

- 2261 • Query identifier: `http://example.com/user`
- 2262 • Asserted CanonicalID: `http://example.com/user#1234`

2263 XRDS (simplified for illustration purposes):

```
2264 <XRDS>
2265 <XRD>
2266 <CanonicalID>http://example.com/user#1234</CanonicalID>
2267 <Service priority="10">
2268 ...
2269 </Service>
2270 ...
2271 </XRD>
2272 </XRDS>
```

2273 The asserted CanonicalID satisfies the HTTP(S) URI verification rules in section 13.1.1.

2274

- 2275 • Query: `=example.name*delegate.name`
- 2276 • CanonicalID: `!=1000.62b1.44fd.2855!1234`

2277 XRDS (for `=example.name*delegate.name`):

```
2278 <XRDS>
2279 <XRD>
2280 <Query>*example.name</Query>
2281 <ProviderID>xri://=</ProviderID>
2282 <LocalID>!1000.62b1.44fd.2855</LocalID>
2283 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2284 <Service>
2285 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2286 <Type>xri://$res*auth*($v*2.0)</Type>
2287 <MediaType>application/xrds+xml</MediaType>
2288 <URI priority="10">http://resolve.example.com</URI>
2289 <URI priority="15">http://resolve2.example.com</URI>
2290 <URI>https://resolve.example.com</URI>
2291 </Service>
2292 ...
2293 </XRD>
2294 <XRD>
2295 <Query>*delegate.name</Query>
2296 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2297 <LocalID>!1234</LocalID>
2298 <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
2299 <Service priority="1">
2300 ...
2301 </Service>
2302 ...
2303 </XRD>
2304 </XRDS>
```

2305 The asserted CanonicalID satisfies the XRI verification rules in section 13.1.2.

2306 13.4.2 CanonicalEquivID Verification

- 2307 • Query identifier: `http://example.com/user`
- 2308 • Asserted CanonicalEquivID: `https://different.example.net/path/user`

2309 First XRDS (for `http://example.com/user`):

```
2310 <XRDS>
2311 <XRD>
2312 <CanonicalID>http://example.com/user</CanonicalID>
2313 <CanonicalEquivID>
2314 https://different.example.net/path/user
2315 </CanonicalEquivID>
2316 <Service priority="10">
2317 ...
2318 </Service>
2319 ...
2320 </XRD>
2321 </XRDS>
```

2322 Second XRDS (for `https://different.example.net/path/user`):

```
2323 <XRDS>
2324 <XRD>
2325 <CanonicalID>https://different.example.net/path/user</CanonicalID>
2326 <EquivID>http://example.com/user</EquivID>
2327 <Service priority="10">
2328 ...
2329 </Service>
2330 ...
2331 </XRD>
2332 </XRDS>
```

2333 The asserted CanonicalEquivID satisfies the verification rules in section 13.2 because it resolves
2334 to a second XRDS that asserts an EquivID whose value is the CanonicalID of the first XRDS.

2335

-
- 2336 • Query identifier: `http://example.com/user`
 - 2337 • Asserted CanonicalEquivID: `!1000.62b1.44fd.2855`

2338 XRDS (for `http://example.com/user`):

```
2339 <XRDS>
2340 <XRD>
2341 <CanonicalID>http://example.com/user</CanonicalID>
2342 <CanonicalEquivID>xri://=!1000.62b1.44fd.2855</CanonicalEquivID>
2343 <Service priority="10">
2344 ...
2345 </Service>
2346 ...
2347 </XRD>
2348 </XRDS>
```

2349 XRDS (for `xri://=!1000.62b1.44fd.2855`):

```
2350 <XRDS>
2351 <XRD>
2352 <Query>!1000.62b1.44fd.2855</Query>
2353 <ProviderID>xri://=!</ProviderID>
2354 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2355 <EquivID>http://example.com/user</EquivID>
```

```
2356 <Service priority="10">
2357   ...
2358 </Service>
2359   ...
2360 </XRD>
2361 </XRDS>
```

2362

- 2363 • Query identifier: =example.name
- 2364 • Asserted CanonicalEquivID: https://example.com/user

2365 First XRDS (for =example.name):

```
2366 <XRDS>
2367 <XRD>
2368   <Query>*example.name</Query>
2369   <ProviderID>xri://=</ProviderID>
2370   <LocalID>!1000.62b1.44fd.2855</LocalID>
2371   <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2372   <CanonicalEquivID>https://example.user</CanonicalEquivID>
2373   <Service priority="10">
2374     ...
2375   </Service>
2376   ...
2377 </XRD>
2378 </XRDS>
```

2379 Second XRDS (for https://example.com/user):

```
2380 <XRDS>
2381 <XRD>
2382   <CanonicalID>https://example.com/user</CanonicalID>
2383   <EquivID>xri://=!1000.62b1.44fd.2855</EquivID>
2384   <Service priority="10">
2385     ...
2386   </Service>
2387   ...
2388 </XRD>
2389 </XRDS>
```

2390

- 2391 • Query identifier: =example.name*delegate.name
- 2392 • Asserted CanonicalEquivID: @!1000.f3da.9056.aca3!5555

2393 First XRDS (for =example.name*delegate.name):

```
2394 <XRDS>
2395 <XRD>
2396   <Query>*example.name</Query>
2397   <ProviderID>xri://=</ProviderID>
2398   <LocalID>!1000.62b1.44fd.2855</LocalID>
2399   <CanonicalID>=!1000.62b1.44fd.2855</CanonicalID>
2400   <Service>
2401     <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2402     <Type>xri://$res*auth*($v*2.0)</Type>
2403     <MediaType>application/xrds+xml</MediaType>
2404     <URI priority="10">http://resolve.example.com</URI>
2405     <URI priority="15">http://resolve2.example.com</URI>
2406     <URI>https://resolve.example.com</URI>
```

```

2407     </Service>
2408     ...
2409 </XRD>
2410 <XRD>
2411   <Query>*delegate.name</Query>
2412   <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2413   <LocalID>!1234</LocalID>
2414   <CanonicalID>=!1000.62b1.44fd.2855!1234</CanonicalID>
2415   <CanonicalEquivID>@!1000.f3da.9056.aca3!5555</CanonicalEquivID>
2416   <Service priority="1">
2417     ...
2418   </Service>
2419   ...
2420 </XRD>
2421 </XRDS>

```

- 2422 • Second XRDS (for @!1000.f3da.9056.aca3!5555):

```

2423 <XRDS>
2424 <XRD>
2425   <Query>!1000.f3da.9056.aca3</Query>
2426   <ProviderID>xri://@</ProviderID>
2427   <GlobalID>@!1000.f3da.9056.aca3</GlobalID>
2428   <CanonicalID>@!1000.f3da.9056.aca3</CanonicalID>
2429   <Service>
2430     <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
2431     <Type>xri://$res*auth*($v*2.0)</Type>
2432     <MediaType>application/xrds+xml</MediaType>
2433     <URI priority="10">http://resolve.example.com</URI>
2434     <URI priority="15">http://resolve2.example.com</URI>
2435     <URI>https://resolve.example.com</URI>
2436   </Service>
2437   ...
2438 </XRD>
2439 <XRD>
2440   <Query>!5555</Query>
2441   <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
2442   <LocalID>!5555</LocalID>
2443   <CanonicalID>xri://@!1000.f3da.9056.aca3!5555</CanonicalID>
2444   <Backref>xri://=!1000.62b1.44fd.2855!1234</Backref>
2445   <Service priority="1">
2446     ...
2447   </Service>
2448   ...
2449 </XRD>
2450 </XRDS>

```

2451 13.5 EquivID Verification

2452 Although XRI resolvers do not automatically perform EquivID synonym verification, a consuming
 2453 application can easily request it using the following steps:

- 2454 1. First request resolution for the original query identifier with `cid=true`.
- 2455 2. From the final XRD in the resolution chain, select the EquivID for which verification is
 2456 desired.
- 2457 3. Request resolution of the EquivID identifier.
- 2458 4. From the final XRD in this second resolution chain, determine if there is either: a) a
 2459 `xrd:XRD/xrd:EquivID` element, or b) a `xrd:XRD/xrd:CanonicalEquivID` element
 2460 whose value matches the verified CanonicalID of the original query identifier. If there is a
 2461 match, the EquivID is verified; otherwise it is not verified.

2462

14 Error Processing

2463

14.1 Error Codes

2464

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

2465

2466

- 1xx: Success—the requested resolution operation was completed successfully.

2467

- 2xx: Permanent errors—the resolver encountered an error from which it could not recover.

2468

- 3xx: Temporary errors—the resolver encountered an error condition that may be only temporary.

2469

2470

Each major category is broken into five minor categories:

2471

- x0x: General error that may take place during any phase of resolution.

2472

- x1x: Input error (or CanonicalID verification status for the 1xx series)

2473

- x2x: Generic authority resolution error.

2474

- x3x: Trusted authority resolution error.

2475

- x4x: Service endpoint (SEP) selection error.

2476

The full list of XRI resolution error codes is defined in Table 26.

2477

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.
110	CID_VERIFIED	Any	CanonicalID verification succeeded. See Table 24.
111	CID_VERIFICATION_FAILED	Any	CanonicalID verification failed. See Table 24.
112	CID_NOT_PRESENT	Any	A CanonicalID element was not present. See Table 24.
113	CEID_VERIFIED	Any	Both CanonicalID and CanonicalEquiVD verification succeeded. See Table 24.
114	CEID_VERIFICATION_FAILED	Any	CanonicalID verification succeeded but CanonicalEquiVD verification failed. See Table 24.
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.

Comment [DSR20]: Do we want to change the name of this symbolic error to INVALID_RES_OUTPUT_FORMAT?

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).

2478 *Table 26: Error codes for XRI resolution.*

2479 As defined in section 7.2, when resolution output is an error and the output format is an XRDS
2480 Document or XRD Document, the error code is returned as the value of the `xrd:code` attribute
2481 of the `xrd:Status` element. When the resolution output is a URI List, the error code is returned
2482 as the first line of a plain text message.

2483 14.2 Error Context Strings

2484 Each error code in Table 26 MAY be returned with an optional error context string that provides
2485 additional human-readable information about the error. When resolution output is an XRDS
2486 Document or XRD Document, this string is returned as the contents of the `xrd:Status` element.
2487 When the resolution output is a URI List, this string MUST be returned as the second line of a
2488 plain text message as specified in section 10.6. Implementers SHOULD provide error context
2489 strings with additional information about an error and possible solutions whenever it can be
2490 helpful to developers or end users.

2491 14.3 Error Handling in Recursing and Proxy Resolution

2492 In recursing and proxy resolution (sections 8.1.6 and 10), a server is acting as a client resolver for
2493 other authority resolution service endpoints. If in this intermediary capacity it receives an
2494 unrecoverable error, it MUST return the error to the originating client in the output format
2495 specified by the value of the requested Resolution Output Format as defined in section 7.2.

2496 If the output format is an XRDS Document, it MUST contain `xrd:XRD` elements for all
2497 subsegments successfully resolved or retrieved from cache prior to the error. The final `xrd:XRD`
2498 element MUST include the `xrd:Query` element that produced the error and the `xrd:Status`
2499 element that describes the error as defined above.

2500 If the output format is an XRD Document, it MUST include the `xrd:Query` element that produced
2501 the error and the `xrd:Status` element that describes the error as defined above.

2502 If this output format is a URI List, it MUST be returned with the content type `text/plain`. The
2503 first line MUST consist of only the numeric error code as defined in section 14.1 followed by a
2504 CRLF. The second line is OPTIONAL; if present it MUST be the error context string as defined in
2505 section 14.2.

2506 If the value of the Resolution Output Format is null (which can only happen in proxy resolution as
2507 described in section 10.5), rather than returning an HTTP(S) redirect, a proxy resolver SHOULD
2508 return a human-readable error message with a media type of either `text/plain` or `text/html`.
2509 It is particularly important in this case to return an error message that will be understandable to an

2510 end-user who may have no understanding of XRI resolution or the fact that the error is coming
2511 from an XRI proxy resolver.

2512 15 Use of HTTP(S)

2513 15.1 HTTP Errors

2514 When a resolver encounters fatal HTTP(S) errors during the resolution process, it **MUST** return
2515 the appropriate XRI resolution error code and error message as defined in section 14. In this way
2516 calling applications do not have to deal separately with XRI and HTTP error messages.

2517 15.2 HTTP Headers

2518 15.2.1 Caching

2519 The HTTP caching capabilities described by **[RFC2616]** should be leveraged for all types of XRI
2520 resolution service. Specifically, implementations **SHOULD** implement the caching model
2521 described in section 13 of **[RFC2616]**, and in particular, the “Expiration Model” of section 13.2, as
2522 this requires the fewest round-trip network connections.

2523 All XRI resolution servers **SHOULD** send the Cache-Control or Expires headers in their
2524 responses per section 13.2 of **[RFC2616]** unless there are overriding security or policy reasons to
2525 omit them.

2526 Note that HTTP Cache headers **SHOULD NOT** conflict with expiration information in an XRD.
2527 That is, the expiration date specified by HTTP caching headers **SHOULD NOT** be later than any
2528 of the expiration dates for any of the `xrd:Expires` elements returned in the HTTP response.
2529 This implies that recursing and proxy resolvers **SHOULD** compute the “soonest” expiration date
2530 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching
2531 headers for the HTTP response.

2532 15.2.2 Location

2533 During HTTP interaction, “Location” headers may be present per **[RFC2616]** (i.e., during 3XX
2534 redirects). Redirects **SHOULD** be made cacheable through appropriate HTTP headers, as
2535 specified in section 15.2.1.

2536 15.2.3 Content-Type

2537 For authority resolution, the “Content-type” header in the 2XX responses **MUST** contain the
2538 media type identifier values specified in Table 11 (for generic resolution), Table 15 (for HTTPS
2539 trusted resolution), Table 16 (for SAML trusted resolution), or Table 17 (or HTTPS+SAML trusted
2540 resolution).

2541 Following service endpoint selection, clients and servers **MAY** negotiate content type using
2542 standard HTTP content negotiation features. Regardless of whether this feature is used,
2543 however, the server **MUST** respond with an appropriate media type in the “Content-type” header
2544 if the resource is found and an appropriate content type is returned.

2545 15.3 Other HTTP Features

2546 HTTP provides a number of other features including transfer-coding, proxying, validation-model
2547 caching, and so forth. All these features may be used insofar as they do not conflict with the
2548 required uses of HTTP described in this document.

2549 15.4 Caching and Efficiency

2550 15.4.1 Resolver Caching

2551 In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the
2552 application level. For best results, however, resolution clients SHOULD be conservative with
2553 caching expiration semantics, including cache expiration dates. This implies that in a series of
2554 HTTP redirects, for example, the results of the entire process SHOULD only be cached as long
2555 as the shortest period of time allowed by any of the intermediate HTTP responses.

2556 Because not all HTTP client libraries expose caching expiration to applications, identifier
2557 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the
2558 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments
2559 should be mindful of limitations in current HTTP clients and proxies.

2560 The cache expiration time of an XRD may also be explicitly limited by the parent authority. If the
2561 expiration time in the `xrd:Expires` element is sooner than the expiration time calculated from
2562 the HTTP caching semantics, the XRD MUST be discarded before the expiration time in
2563 `xrd:Expires`. Note also that a `saml:Assertion` element returned during SAML trusted
2564 resolution has its own signature expiration semantics as defined in [SAML]. While this may
2565 invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if
2566 it is not expired by HTTP caching semantics or the `xrd:Expires` element.

2567 With both application-level and HTTP-level caching, the resolution process is designed to have
2568 minimal overhead. Resolution of each qualified subsegment of an XRI authority segment is a
2569 separate step described by a separate XRD, so intermediate results can typically be cached in
2570 their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified
2571 subsegments, which are common to more identifiers, will naturally result in a greater number of
2572 cache hits than resolution of lower-level subsegments.

2573 15.4.2 Synonyms

2574 The publication of synonyms in XRD documents can further increase cache efficiency. If an XRI
2575 resolution request produces a cache hit on a synonym, the following rules apply:

- 2576 1. If the cache hit is on a LocalID synonym, the resolver MAY return the cached XRD
2577 document if: a) it is from the correct ProviderID, b) it has not expired, and c) it was
2578 obtained using the same trusted resolution and synonym verification parameters as the
2579 current resolution request.
- 2580 2. If the cache hit is on a GlobalID synonym, the resolver MAY return the entire cached
2581 XRDS document if: a) it has not expired, and b) it was obtained using the same trusted
2582 resolution and synonym verification parameters as the current resolution request.

2583 IMPORTANT: The effect of these rules is that the application calling an XRI resolver MAY receive
2584 back an XRD document, or an XRDS document containing XRD document(s), in which the
2585 `<Query>` element does not match the resolution request, but in which a `<LocalID>` element
2586 match does match the resolution request.

2587

16 Extensibility and Versioning

2588

16.1 Extensibility

2589

16.1.1 Extensibility of XRDs

2590
2591
2592
2593
2594
2595
2596

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

2597
2598
2599

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

2600
2601
2602
2603
2604

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

2605
2606
2607
2608
2609

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

2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620

```
<XRD>
  <Service>
    ...
  </Service>
  <other:SuperService>
    <Service>
      ...
      <other:ExtensionElement>...</other:ExtensionElement>
    </Service>
  </other:SuperService>
</XRD>
```

2621
2622
2623
2624
2625
2626

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

2627
2628

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

2629

16.1.2 Other Points of Extensibility

2630
2631

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

- 2632 • Specification of new resolution service types or other service types using XRI or URIs as
2633 values of the `xrd:Type` element.
- 2634 • Specification of new resolution output formats or features using media types and media type
2635 parameters as values of the `xrd:MediaType` element as defined in [RFC2045] and
2636 [RFC2046].
- 2637 • HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- 2638 • Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- 2639 • Use of cross-references within XRIs, particularly for associating new types of metadata with a
2640 resource. See [XRISyntax] and [XRI_Metadata].

2641 16.2 Versioning

2642 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,
2643 this section describes versioning guidelines.

2644 16.2.1 Version Numbering

2645 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number
2646 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version
2647 number *MajorA.MinorA* if and only if:

2648 $Major_B > Major_A$ OR (($Major_B = Major_A$) AND $Minor_B > Minor_A$)

2649 16.2.2 Versioning of the XRI Resolution Specification

2650 New releases of the XRI Resolution specification may specify changes to the resolution protocols
2651 and/or the XRD schema in Appendix A. When changes affect either of these, the resolution
2652 service type version number will be changed. Where changes are purely editorial, the version
2653 number will not be changed.

2654 In general, if a change is backward-compatible, the new version will be identified using the
2655 current major version number and a new minor version number. If the change is not backward-
2656 compatible, the new version will be identified with a new major version number.

2657 16.2.3 Versioning of XRDs

2658 The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have
2659 no specific knowledge of the elements it may contain. Therefore it has no version element, and
2660 can remain stable indefinitely because there is no need to version its namespace.

2661 The `xrd:XRD` element has an optional `xrd:version` attribute. When used, the value of this
2662 attribute MUST be the exact numeric version value of the XRI Resolution specification to which its
2663 containing elements conform.

2664 When new versions of the XRI Resolution specification are released, the namespace for the XRD
2665 schema may or may not be changed. If there is a major version number change, the namespace
2666 for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the
2667 namespace for the `xrd:XRD` schema may remain unchanged.

2668 With regard to versioning, this specification follows the same guidelines as established in section
2669 4.2.1 of [SAML]:

2670 *In general, maintaining namespace stability while adding or changing the content of a*
2671 *schema are competing goals. While certain design strategies can facilitate such changes,*
2672 *it is complex to predict how older implementations will react to any given change, making*
2673 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*
2674 *minor revisions is reserved, in the interest of namespace stability. Except in special*

2675 *circumstances (for example, to correct major deficiencies or to fix errors),*
2676 *implementations should expect forward-compatible schema changes in minor revisions,*
2677 *allowing new messages to validate against older schemas.*
2678 *Implementations SHOULD expect and be prepared to deal with new extensions and*
2679 *message types in accordance with the processing rules laid out for those types. Minor*
2680 *revisions MAY introduce new types that leverage the extension facilities described in [this*
2681 *section]. Older implementations SHOULD reject such extensions gracefully when they*
2682 *are encountered in contexts that dictate mandatory semantics.*

2683 **16.2.4 Versioning of Protocols**

2684 The protocols defined in this document may also be versioned by future releases of the XRI
2685 Resolution specification. If these protocols are not backward-compatible with older
2686 implementations, they will be assigned a new XRI with a new version identifier for use in
2687 identifying their service type in XRDs. See section 3.1.2.

2688 Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP
2689 provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an
2690 XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely
2691 to continue to use the same XRI to identify the protocol as was used in previous versions of the
2692 XRI Resolution specification.

2693

17 Security and Data Protection

2694
2695
2696
2697

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.

2698

17.1 DNS Spoofing or Poisoning

2699
2700
2701
2702
2703
2704
2705
2706
2707

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.

2708

17.2 HTTP Security

2709
2710
2711
2712

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.

2713
2714
2715
2716

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.

2717
2718

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

2719

17.3 SAML Considerations

2720
2721
2722

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.

2723

17.4 Limitations of Trusted Resolution

2724
2725
2726
2727
2728

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.

2729
2730
2731

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.

2732 **17.5 Community Root Authorities**

2733 The XRI authority information for a community root needs to be well-known to the clients that
2734 request resolution within that community. For trusted resolution, this includes the authority
2735 resolution service endpoint URIs, the `xrd:XRD/xrd:ProviderID`, and the `ds:KeyInfo`
2736 information. An acceptable means of providing this information is for the community root authority
2737 to produce a self-signed XRD and publish it to a server-authenticated HTTPS endpoint. Special
2738 care should be taken to ensure the correctness of such an XRD; if this information is incorrect, an
2739 attacker may be able to convince a client of an incorrect result during trusted resolution.

2740 **17.6 Caching Authorities**

2741 In addition to traditional HTTP caching proxies, XRI proxy resolvers may be a part of the
2742 resolution topology. Such proxy resolvers should take special precautions against cache
2743 poisoning, as these caching entities may represent trusted decision points within a deployment's
2744 resolution architecture.

2745 **17.7 Recursing and Proxy Resolution**

2746 During recursing resolution, subsegments of the XRI authority segment for which the resolving
2747 network endpoint is not authoritative may be revealed to that service endpoint. During proxy
2748 resolution, some or all of an XRI is provided to the proxy resolver.

2749 In both cases, privacy considerations should be evaluated before disclosing such information.

2750 **17.8 Denial-Of-Service Attacks**

2751 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks
2752 typical of systems relying on DNS and HTTP.

2753

A. Acknowledgments

2754

The editors would like to acknowledge the contributions of the OASIS XRI Technical Committee,

2755

whose voting members at the time of publication were:

Comment [DSR21]: TO DO - Needs updating.

2756

- Geoffrey Strongin, Advanced Micro Devices

2757

- Ajay Madhok, AmSoft Systems

2758

- Jean-Jacques Dubray, Attachmate

2759

- William Barnhill, Booz Allen and Hamilton

2760

- Drummond Reed, Cordance Corporation

2761

- Marc Le Maitre, Cordance Corporation

2762

- Dave McAlpin, Epok

2763

- Loren West, Epok

2764

- Peter Davis, NeuStar

2765

- Masaki Nishitani, Nomura Research

2766

- Nat Sakimura, Nomura Research

2767

- Tetsu Watanabe, Nomura Research

2768

- Owen Davis, PlaNetwork

2769

- Victor Grey, PlaNetwork

2770

- Fen Labalme, PlaNetwork

2771

- Mike Lindelsee, Visa International

2772

- Gabriel Wachob, Visa International

2773

- Dave Wentker, Visa International

2774

- Bill Washburn, XDI.ORG

2775

The editors also would like to acknowledge the following people for their contributions to previous

2776

versions of the OASIS XRI specifications (affiliations listed for OASIS members):

2777

Thomas Bikeev, EAN International; Krishna Sankar, Cisco; Winston Bumpus, Dell; Joseph

2778

Moeller, EDS; Steve Green, Epok; Lance Hood, Epok; Adarbad Master, Epok; Davis McPherson,

2779

Epok; Chetan Sabnis, Epok; Phillipe LeBlanc, GemPlus; Jim Schreckengast, Gemplus; Xavier

2780

Serret, Gemplus; John McGarvey, IBM; Reva Modi, Infosys; Krishnan Rajagopalan, Novell;

2781

Tomonori Seki, NRI; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,

2782

Verisign; Rajeev Maria, Visa International; Terence Spielman, Visa International; John Veizades,

2783

Visa International; Lark Allen, Wave Systems; Michael Willett, Wave Systems; Matthew Dovey;

2784

Eamonn Neylon; Mary Nishikawa; Lars Marius Garshol; Norman Paskin; Bernard Vatant.

2785

- A special acknowledgement to Jerry Kindall (Epok) for a full editorial review.

B. Non-Normative Text

2787

C. Revision History

2788
2789

[optional; should not be included in OASIS Standards]

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>

2790
2791

D. XML Schema for XRDS and XRD (Normative)

```

2793 <?xml version="1.0" encoding="UTF-8"?>
2794 <xs:schema targetNamespace="xri://$xrds" elementFormDefault="qualified"
2795 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds">
2796   <!-- Utility patterns -->
2797   <xs:attributeGroup name="otherattribute">
2798     <xs:anyAttribute namespace="##other" processContents="lax"/>
2799   </xs:attributeGroup>
2800   <xs:group name="otherelement">
2801     <xs:choice>
2802       <xs:any namespace="##other" processContents="lax"/>
2803       <xs:any namespace="##local" processContents="lax"/>
2804     </xs:choice>
2805   </xs:group>
2806   <!-- Patterns for elements -->
2807   <xs:element name="XRDS">
2808     <xs:complexType>
2809       <xs:sequence>
2810         <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2811       </xs:sequence>
2812       <xs:attributeGroup ref="xrds:otherattribute"/>
2813       <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
2814     </xs:complexType>
2815   </xs:element>
2816 </xs:schema>
2817
2818
2819 <?xml version="1.0" encoding="UTF-8"?>
2820 <xs:schema targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified"
2821 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2822 xmlns:xrd="xri://$xrd*($v*2.0)">
2823   <!-- Utility patterns -->
2824   <xs:attributeGroup name="otherattribute">
2825     <xs:anyAttribute namespace="##other" processContents="lax"/>
2826   </xs:attributeGroup>
2827   <xs:group name="otherelement">
2828     <xs:choice>
2829       <xs:any namespace="##other" processContents="lax"/>
2830       <xs:any namespace="##local" processContents="lax"/>
2831     </xs:choice>
2832   </xs:group>
2833   <xs:attributeGroup name="priorityAttrGrp">
2834     <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
2835   </xs:attributeGroup>
2836   <xs:attributeGroup name="selectionAttrGrp">
2837     <xs:attribute name="match" use="optional" default="default">
2838       <xs:simpleType>
2839         <xs:restriction base="xs:string">
2840           <xs:enumeration value="default"/>
2841           <xs:enumeration value="content"/>
2842           <xs:enumeration value="any"/>
2843           <xs:enumeration value="non-null"/>
2844           <xs:enumeration value="null"/>
2845           <xs:enumeration value="none"/>
2846         </xs:restriction>
2847       </xs:simpleType>
2848     </xs:attribute>
2849     <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
2850   </xs:attributeGroup>
2851   <xs:complexType name="URIPattern">
2852     <xs:simpleContent>
2853       <xs:extension base="xs:anyURI">
2854         <xs:attributeGroup ref="xrd:otherattribute"/>
2855       </xs:extension>
2856     </xs:simpleContent>
2857   </xs:complexType>
2858   <xs:complexType name="URIPriorityPattern">

```

```

2859     <xs:simpleContent>
2860         <xs:extension base="xrd:URIPattern">
2861             <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2862         </xs:extension>
2863     </xs:simpleContent>
2864 </xs:complexType>
2865 <xs:complexType name="StringPattern">
2866     <xs:simpleContent>
2867         <xs:extension base="xs:string">
2868             <xs:attributeGroup ref="xrd:otherattribute"/>
2869         </xs:extension>
2870     </xs:simpleContent>
2871 </xs:complexType>
2872 <xs:complexType name="StringSelectionPattern">
2873     <xs:simpleContent>
2874         <xs:extension base="xrd:StringPattern">
2875             <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
2876         </xs:extension>
2877     </xs:simpleContent>
2878 </xs:complexType>
2879 <!-- Patterns for elements -->
2880 <xs:element name="XRD">
2881     <xs:complexType>
2882         <xs:sequence>
2883             <xs:element ref="xrd:Query" minOccurs="0"/>
2884             <xs:element ref="xrd:Status" minOccurs="0"/>
2885             <xs:element ref="xrd:Expires" minOccurs="0"/>
2886             <xs:element ref="xrd:ProviderID" minOccurs="0"/>
2887             <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
2888             <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded"/>
2889             <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
2890             <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded"/>
2891             <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2892         </xs:sequence>
2893         <xs:attribute name="id" type="xs:ID"/>
2894         <xs:attribute name="idref" type="xs:IDREF" use="optional"/>
2895         <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0"/>
2896         <xs:attributeGroup ref="xrd:otherattribute"/>
2897     </xs:complexType>
2898 </xs:element>
2899 <xs:element name="Query" type="xrd:StringPattern"/>
2900 <xs:element name="Status">
2901     <xs:complexType>
2902         <xs:simpleContent>
2903             <xs:extension base="xrd:StringPattern">
2904                 <xs:attribute name="code" type="xs:int" use="required"/>
2905                 <xs:attributeGroup ref="xrd:otherattribute"/>
2906             </xs:extension>
2907         </xs:simpleContent>
2908     </xs:complexType>
2909 </xs:element>
2910 <xs:element name="Expires">
2911     <xs:complexType>
2912         <xs:simpleContent>
2913             <xs:extension base="xs:dateTime">
2914                 <xs:attributeGroup ref="xrd:otherattribute"/>
2915             </xs:extension>
2916         </xs:simpleContent>
2917     </xs:complexType>
2918 </xs:element>
2919 <xs:element name="ProviderID" type="xrd:URIPattern"/>
2920 <xs:element name="LocalID">
2921     <xs:complexType>
2922         <xs:simpleContent>
2923             <xs:extension base="xrd:StringPattern">
2924                 <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2925             </xs:extension>
2926         </xs:simpleContent>
2927     </xs:complexType>
2928 </xs:element>
2929 <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>

```

```

2930 <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
2931 <xs:element name="Service">
2932   <xs:complexType>
2933     <xs:sequence>
2934       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
2935       <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
2936       <xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
2937       <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
2938       <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
2939       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
2940     </xs:sequence>
2941     <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2942     <xs:attributeGroup ref="xrd:otherattribute"/>
2943   </xs:complexType>
2944 </xs:element>
2945 <xs:element name="Type">
2946   <xs:complexType>
2947     <xs:simpleContent>
2948       <xs:extension base="xrd:URIPattern">
2949         <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
2950       </xs:extension>
2951     </xs:simpleContent>
2952   </xs:complexType>
2953 </xs:element>
2954 <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
2955 <xs:element name="Path" type="xrd:StringSelectionPattern"/>
2956 <xs:element name="URI">
2957   <xs:complexType>
2958     <xs:simpleContent>
2959       <xs:extension base="xrd:URIPattern">
2960         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
2961         <xs:attribute name="append">
2962           <xs:simpleType>
2963             <xs:restriction base="xs:string">
2964               <xs:enumeration value="none"/>
2965               <xs:enumeration value="local"/>
2966               <xs:enumeration value="authority"/>
2967               <xs:enumeration value="path"/>
2968               <xs:enumeration value="query"/>
2969               <xs:enumeration value="qxri"/>
2970             </xs:restriction>
2971           </xs:simpleType>
2972         </xs:attribute>
2973       </xs:extension>
2974     </xs:simpleContent>
2975   </xs:complexType>
2976 </xs:element>
2977 </xs:schema>
2978
2979

```

2980

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

2981

2982 [TODO]

2983 **F. Media Type Definition for application/xrds+xml**
2984 **(Normative)**

2985 This section is prepared in anticipation of media type registration meeting the requirements of
2986 **[RFC4288]**.

2987 **Type name:** application

2988 **Subtype name:** xrds+xml

2989 **Required parameters:** None

2990 **Optional parameters:** See Table 6 of this document.

2991 **Encoding considerations:** Identical to those of "application/xml" as described in **[RFC3023]**,
2992 Section 3.2.

2993 **Security considerations:** As defined in this specification. In addition, as this media type uses the
2994 "+xml" convention, it shares the same security considerations as described in **[RFC3023]**,
2995 Section 10.

2996 **Interoperability considerations:** There are no known interoperability issues.

2997 **Published specification:** This specification.

2998 **Applications that use this media type:** Applications conforming to this specification use this
2999 media type.

3000 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
3001 Technical Committee Co-Chair, drummond.reed@cordance.net

3002 **Intended usage:** COMMON

3003 **Restrictions on usage:** None

3004 **Author:** OASIS XRI TC

3005 **Change controller:** OASIS XRI TC

3006 **G. Media Type Definition for application/xrd+xml**
3007 **(Normative)**

3008 This section is prepared in anticipation of media type registration meeting the requirements of
3009 **[RFC4288]**.

3010 **Type name:** application

3011 **Subtype name:** xrd+xml

3012 **Required parameters:** None

3013 **Optional parameters:** See Table 6 of this document.

3014 **Encoding considerations:** Identical to those of "application/xml" as described in **[RFC3023]**,
3015 Section 3.2.

3016 **Security considerations:** As defined in this specification. In addition, as this media type uses the
3017 "+xml" convention, it shares the same security considerations as described in **[RFC3023]**,
3018 Section 10.

3019 **Interoperability considerations:** There are no known interoperability issues.

3020 **Published specification:** This specification.

3021 **Applications that use this media type:** Applications conforming to this specification use this
3022 media type.

3023 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
3024 Technical Committee Co-Chair, drummond.reed@cordance.net

3025 **Intended usage:** COMMON

3026 **Restrictions on usage:** None

3027 **Author:** OASIS XRI TC

3028 **Change controller:** OASIS XRI TC

3029
3030

H. Example Local Resolver Interface Definition (Informative)

Comment [DSR23]: Needs updating (issue #38 and new parameter types)

3031
3032
3033
3034
3035
3036
3037

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

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 7.1.1.
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 7.1.2.
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 7.1.2.
sepType	Service Type as defined in section 7.1.3.
sepMediaType	Service Media Type as defined in section 7.1.4.

Comment [DSR24]: Needs updating

3038
3039
3040
3041

The five operations correspond to the following combinations of values of the Resolution Output Format parameter and its `sep` (service endpoint) subparameter (section 7.1.2) as shown below.

	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

3042 Following is the API and descriptions of the five operations.

3043 1. Resolve Authority to XRDS

```
3044 int resolveAuthToXRDS(  
3045     in string QXRI, in string trustType, in boolean followRefs,  
3046     out string XRDS, out string errorContext);
```

- 3047 • Performs authority resolution only (sections 5 and 6) and outputs the XRDS as specified in
3048 section 4.2.1 when the `sep` subparameter is `false`.
- 3049 • Only the authority segment of the QXRI is processed by this function. If the QXRI contains a
3050 path or query component, it is ignored.
- 3051 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 3052 • Returns the error code. If error, then the `errorContext` output argument may contain additional
3053 error information. The output XRDS will contain a final XRD with the same status code and
3054 optional context information in its `xrd:Status` element.

3055

3056 2. Resolve Authority to XRD

```
3057 int resolveAuthToXRD(  
3058     in string QXRI, in string trustType, in boolean followRefs,  
3059     out string XRD, out string errorContext);
```

- 3060 • Performs authority resolution only (sections 5 and 6) and outputs the final XRD as specified
3061 in section 4.2.2 when the `sep` subparameter is `false`.
- 3062 • Only the authority segment of the QXRI is processed by this function. If the QXRI contains a
3063 path or query component, it is ignored.
- 3064 • The output XRD argument will be signed or not depending on the value of `trustType`.
- 3065 • Returns the error code. If error, then the `errorContext` output argument may contain
3066 additional error information. The output XRD will contain the same status code and optional
3067 context information in its `xrd:Status` element.

3068

3069 3. Resolve Service Endpoint to XRDS

```
3070 int resolveSEPToXRDS(  
3071     in string QXRI, in string trustType, in string sepType,  
3072     in string sepMediaType, in boolean followRefs,  
3073     out string XRDS, out string errorContext);
```

- 3074 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
3075 and outputs the XRDS as specified in section 4.2.1 when the `sep` subparameter is `true`.
- 3076 • As specified in section 4.2.1, the output XRDS document will contain a nested XRDS
3077 document as the final child element if reference processing was necessary to locate the
3078 request service endpoint and the `followRefs` flag was set to `true`.
- 3079 • The final XRD in the output XRD will either contain at least one instance of the requested
3080 service endpoint or an error. IMPORTANT: Although the resolver will perform this verification,
3081 the final XRD is NOT filtered when the Resolution Output Format is an XRDS document.
3082 Filtering is only performed when the Resolution Output Format is an XRD document (see
3083 next section).
- 3084 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 3085 • Returns the error code. If error, then the `errorContext` output argument may contain
3086 additional error information. The output XRDS will contain a final XRD with the same status
3087 code and optional context information in its `xrd:Status` element.
- 3088 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
3089 same as the value `null`.

3090

3091 4. Resolve Service Endpoint to XRD

```
3092 int resolveSEPToXRD(  
3093     in string QXRI, in string trustType, in string sepType,  
3094     in string sepMediaType, in boolean followRefs,  
3095     out string XRDS, out string errorContext);
```

- 3096 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
3097 and outputs the XRD as specified in section 4.2.2 when the `sep` subparameter is `true`.
- 3098 • As specified in section 4.2.2, all elements in the output XRD subject to the global
3099 `xrd:priority` attribute will be returned in order of highest to lowest.
- 3100 • The output XRD will either contain at least one instance of the requested service endpoint or
3101 an error.
- 3102 • The output XRD will be *not* be signed regardless of the value of `trustType` because the
3103 XRD will be filtered to only the selected service endpoints.
- 3104 • Returns the error code. If error, then the `errorContext` output argument may contain
3105 additional error information. The output XRD will contain the same status code and optional
3106 context information in its `xrd:Status` element.
- 3107 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
3108 same as the value `null`.

3109 **5. Resolve Service Endpoint to URI List**

```
3110 int resolveSepToURIList(  
3111     in string QXRI, in string trustType, in string sepType,  
3112     in string sepMediaType, in boolean followRefs,  
3113     out string[] URIList, out string errorContext);
```

- 3114 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
3115 and, upon success, outputs a non-empty URI List as specified in section 4.2.3.
- 3116 • Returns the error code. If error, then the output URI List will be empty, and the
3117 `errorContext` output argument may contain additional error information.
- 3118 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
3119 same as the value null.