



Extensible Resource Identifier (XRI) Resolution Version 2.0

Working Draft 11, ED 06

16 October 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 a generic format for resource description (XRDS documents), a protocol for obtaining XRDS documents from HTTP(S) URIs, and generic and trusted protocols for resolving Extensible Resource Identifiers (XRIs) using XRDS documents. These protocols are intended for use both with HTTP(S) URIs and with XRIs as defined by *Extensible Resource Identifier (XRI) Syntax V2.0* [XRISyntax] or higher. For a dictionary of XRIs defined to provide standardized identifier metadata, see *Extensible Resource Identifier (XRI) Metadata V2.0* [XRIMetadata]. For a basic introduction to XRIs, see the *XRI 2.0 FAQ* [XRIFAQ]. For an implementers guide, see the *XRI Resolution 2.0 Implementers Guide* [XRIGuide].

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 [DSR6]: TODO – Need to ensure all statements are 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	10
1.1	Overview of XRI Resolution Architecture	10
1.2	Structure of this Specification	13
1.3	Examples of XRI Resolution Requests and Responses	14
1.4	Terminology and Notation	14
1.5	Normative References	15
1.6	Non-Normative References	16
2	Conformance	17
2.1	Conformance Targets	17
2.2	XRDS Clients	17
2.3	XRDS Servers	17
2.4	XRI Resolvers	17
2.4.1	Local Resolvers	17
2.4.2	Proxy Resolvers	17
2.5	XRI Authority Servers	17
3	Namespaces	18
3.1	XRI Namespaces for XRI Resolution	18
3.1.1	XRIs Reserved for XRI Resolution	18
3.1.2	XRIs Assigned to XRI Resolution Service Types	18
3.2	XML Namespaces for XRI Resolution	18
3.3	Media Types for XRI Resolution	19
4	XRDS Documents	21
4.1	XRDS and XRD Namespaces	21
4.2	XRD Elements and Attributes	21
4.2.1	Management Elements	23
4.2.2	Trust Elements	24
4.2.3	Synonym Elements	24
4.2.4	Service Endpoint Descriptor Elements	25
4.2.5	Service Endpoint Trust Elements	26
4.2.6	Service Endpoint Selection Elements	26
4.3	XRD Attribute Processing Rules	27
4.3.1	ID Attribute	27
4.3.2	Version Attribute	27
4.3.3	Priority Attribute	27
4.4	XRI and IRI Encoding Requirements	28
5	XRDS Synonyms	29
5.1	Query Identifiers	29
5.1.1	HTTP(S) URI Query Identifiers	29
5.1.2	XRI Query Identifiers	29
5.2	XRD Synonym Elements	30
5.2.1	LocalID	30
5.2.2	EquivID	30

5.2.3 CanonicalID	31
5.2.4 CanonicalEquivID	31
5.3 Redirect and Ref Elements	32
5.4 Synonym Verification.....	32
5.5 Synonym Selection	32
6 Discovering an XRDS Document from an HTTP(S) URI	34
6.1 Overview	34
6.2 HEAD Protocol.....	34
6.3 GET Protocol.....	34
7 XRI Resolution Inputs and Outputs	36
7.1 Inputs	37
7.1.1 QXRI (Authority String, Path String, and Query String)	39
7.1.2 Resolution Output Format	39
7.1.3 Service Type	40
7.1.4 Service Media Type	40
7.2 Outputs	42
7.2.1 XRDS Document.....	44
7.2.2 XRD Document	44
7.2.3 URI List	45
7.2.4 HTTP(S) Redirect	45
8 Generic Authority Resolution	46
8.1 XRI Authority Resolution	46
8.1.1 Service Type and Service Media Type.....	46
8.1.2 Protocol.....	47
8.1.3 Requesting an XRDS Document using HTTP(S)	48
8.1.4 Failover Handling	49
8.1.5 Community Root Authorities	50
8.1.6 Self-Describing XRDS Documents.....	51
8.1.7 Qualified Subsegments.....	51
8.1.8 Cross-References	52
8.1.9 Selection of the Next Authority Resolution Service Endpoint	52
8.1.10 Construction of the Next Authority URI.....	53
8.1.11 Recursing Authority Resolution	53
8.2 IRI Authority Resolution.....	54
8.2.1 Service Type and Media Type	54
8.2.2 Protocol.....	54
8.2.3 Optional Use of HTTPS.....	54
9 Trusted Authority Resolution.....	55
9.1 HTTPS	55
9.1.1 Service Type and Service Media Type.....	55
9.1.2 Protocol.....	55
9.2 SAML	55
9.2.1 Service Type and Service Media Type.....	56
9.2.2 Protocol.....	56
9.2.3 Recursing Authority Resolution	57

9.2.4 Client Validation of XRDS.....	57
9.2.5 Correlation of ProviderID and KeyInfo Elements.....	58
9.3 HTTPS+SAML.....	59
9.3.1 Service Type and Service Media Type.....	59
9.3.2 Protocol.....	59
10 Proxy Resolution.....	60
10.1 Service Type and Media Types.....	60
10.2 HXRIs.....	60
10.3 HXRI Query Parameters.....	62
10.4 HTTP(S) Accept Headers.....	63
10.5 Null Resolution Output Format.....	63
10.6 Outputs and HTTP(S) Redirects.....	63
10.7 Differences Between Proxy Resolution Servers.....	64
10.8 Combining Authority and Proxy Resolution Servers.....	64
11 Redirect and Ref Processing.....	65
11.1 Cardinality.....	66
11.2 Redirect Processing.....	67
11.3 Ref Processing.....	68
11.4 Nested XRDS Documents.....	69
11.4.1 Redirect Examples.....	70
11.4.2 Ref Examples.....	72
11.5 Recursion and Backtracking.....	74
12 Service Endpoint Selection.....	76
12.1 Processing Rules.....	76
12.2 Service Endpoint Selection Logic.....	78
12.3 Selection Element Matching Rules.....	78
12.3.1 Selection Element Match Options.....	79
12.3.2 The Match Attribute.....	79
12.3.3 Absent Selection Element Matching Rule.....	79
12.3.4 Empty Selection Element Matching Rule.....	80
12.3.5 Multiple Selection Element Matching Rule.....	80
12.3.6 Type Element Matching Rules.....	80
12.3.7 Path Element Matching Rules.....	80
12.3.8 MediaType Element Matching Rules.....	82
12.4 Service Endpoint Matching Rules.....	82
12.4.1 Service Endpoint Match Options.....	82
12.4.2 Select Attribute Match Rule.....	82
12.4.3 All Positive Match Rule.....	82
12.4.4 Default Match Rule.....	82
12.5 Service Endpoint Selection Rules.....	83
12.5.1 Positive Match Rule.....	83
12.5.2 Default Match Rule.....	83
12.6 Pseudocode.....	83
12.7 Construction of Service Endpoint URIs.....	85
12.7.1 The uric Parameter.....	85

12.7.2 The Append Attribute	85
13 Synonym Verification	87
13.1 EquivID Verification	87
13.2 CanonicalID Verification	88
13.2.1 HTTP(S) URI Verification Rules	88
13.2.2 XRI Verification Rules	88
13.2.3 CanonicalEquivID Verification	89
13.2.4 Verification Status Attributes	89
13.2.5 Examples	90
14 Status Codes and Error Processing	95
14.1 Status Elements	95
14.2 Status Codes	95
14.3 Status Context Strings	98
14.4 Returning Errors in Plain Text or HTML	98
14.5 Error Handling in Recursing and Proxy Resolution	98
15 Use of HTTP(S)	99
15.1 HTTP Errors	99
15.2 HTTP Headers	99
15.2.1 Caching	99
15.2.2 Location	99
15.2.3 Content-Type	99
15.3 Other HTTP Features	99
15.4 Caching and Efficiency	100
15.4.1 Resolver Caching	100
15.4.2 Synonyms	100
16 Extensibility and Versioning	101
16.1 Extensibility	101
16.1.1 Extensibility of XRDs	101
16.1.2 Other Points of Extensibility	101
16.2 Versioning	102
16.2.1 Version Numbering	102
16.2.2 Versioning of the XRI Resolution Specification	102
16.2.3 Versioning of Protocols	102
16.2.4 Versioning of XRDs	103
17 Security and Data Protection	104
17.1 DNS Spoofing or Poisoning	104
17.2 HTTP Security	104
17.3 SAML Considerations	104
17.4 Limitations of Trusted Resolution	104
17.5 Synonym Verification	105
17.6 Redirect and Ref Management	105
17.7 Community Root Authorities	105
17.8 Caching Authorities	105
17.9 Recursing and Proxy Resolution	105
17.10 Denial-Of-Service Attacks	105

A. Acknowledgments.....	106
B. Non-Normative Text.....	107
C. Revision History.....	108
D. XML Schema for XRDS and XRD (Normative)	109
E. RelaxNG Compact Syntax Schema for XRDS and XRD (Informative).....	112
F. Media Type Definition for application/xrds+xml (Normative)	113
G. Media Type Definition for application/xrd+xml (Normative).....	114
H. Example Local Resolver Interface Definition (Informative).....	115

Table of Figures

Figure 1: Four typical scenarios for XRI authority resolution.....	12
Figure 2: Top-level flowchart of XRI resolution phases.....	36
Figure 3: Input processing flowchart.....	38
Figure 4: Output processing flowchart.....	43
Figure 5: Authority resolution flowchart.....	47
Figure 6: XRDS request flowchart.....	48
Figure 7: Redirect and Ref processing flowchart.....	66
Figure 8: Service endpoint selection flowchart.....	76
Figure 9: Service endpoint (SEP) selection logic flowchart.....	78

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 [simple generic resource description format called XRDS \(Extensible Resource Descriptor Sequence\), a standard protocol for resolving HTTP\(S\) URIs to obtain XRDS documents, and standard protocol for resolving XRIs using XRDS documents](#). Both generic and trusted versions [of the XRI resolution protocol](#) 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.

Deleted: HTTP(S)

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 or 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.

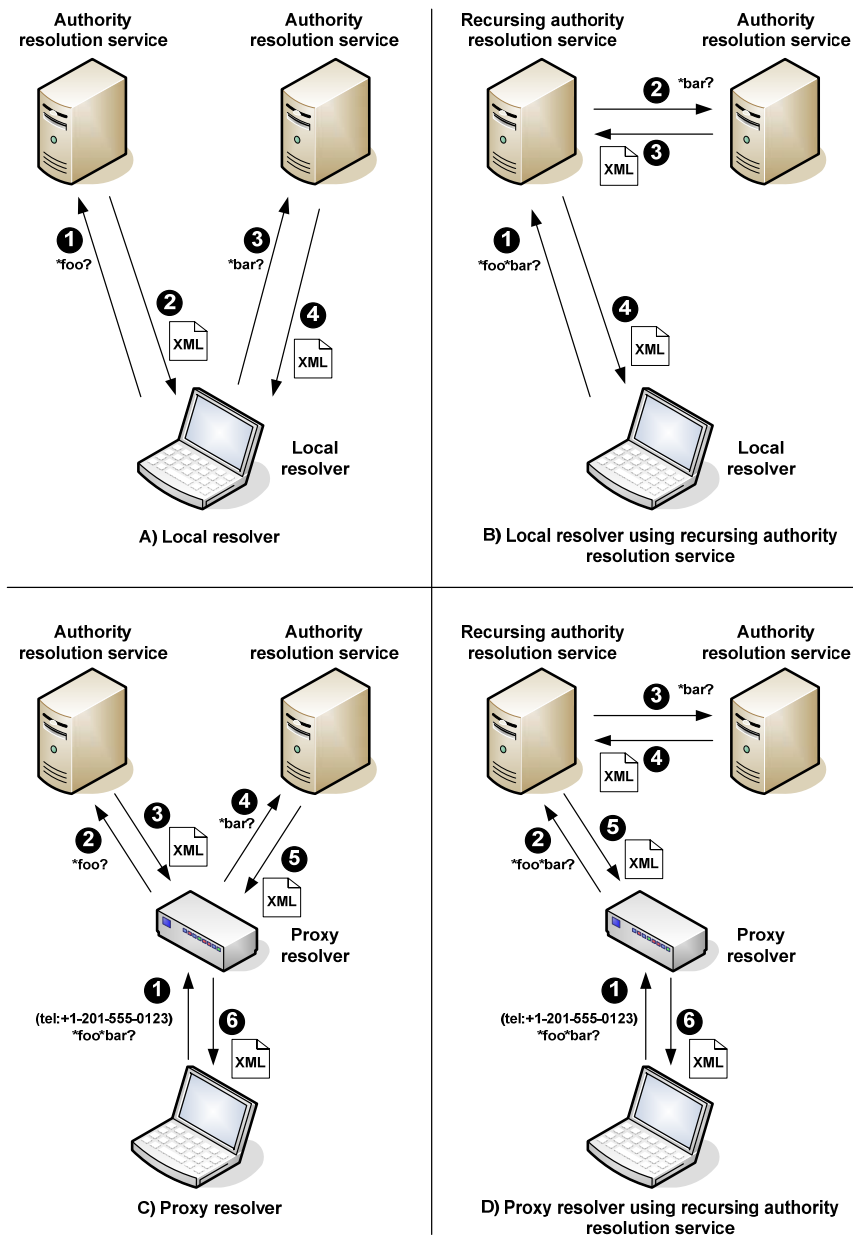
Resolution Component	DNS Architecture	XRI Architecture
Identifier	domain name	XRI (authority + path + query)
Resource record format	text (resource record)	XML (XRDS document)
Attribute identifier	string	anyURI
Network endpoint identifier	IP address	URI
Synonyms	CNAME	LocalID, EquivID, CanonicalID, CanonicalEquivID
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	recursing nameserver	recursing authority resolution

		service or proxy resolver
--	--	---------------------------

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

32 As Table 1 notes, XRI resolution architecture supports both recursing authority resolution
33 services and *proxy resolvers*. A proxy resolver is simply an HTTP(S) interface to a local XRI
34 resolver (one implemented using a platform-specific API). Proxy resolvers enable applications—
35 even those that do not natively understand XRIs but can process HTTP URIs—to easily access
36 the functions of an XRI resolver remotely.

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



40

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

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

- 43 • *Phase 1: Authority Resolution.* This is the phase required to resolve the authority segment of
 44 an XRI into an XRDS document describing the target authority. Authority resolution works
 45 iteratively from left-to-right across each subsegment in the authority segment of the XRI. In
 46 XRIs, subsegments are delimited using either a specified set of symbol characters or
 47 parentheses. For example, in the XRI `xri://(http://example.root)*foo*bar`, the

48 authority subsegments are (`http://example.root`) (the community root authority, in this
49 case a URI expressed as an cross-reference delimited with parentheses), `*foo`, (the first
50 resolvable subsegment), and `*bar`, (the second resolvable subsegment). Note that a
51 resolver must be preconfigured (or have its own way of discovering) the community root
52 authority starting point, so the community root subsegment is not resolved except in one
53 special case (see section 8.1.6).

54 • *Phase 2: Service Endpoint Selection.* Once authority resolution is complete, the optional
55 second phase of XRI resolution is to select a specific set of metadata from the final XRDS
56 document retrieved. Although an XRDS document may contain any type of metadata
57 describing the target resource, this specification defines a ruleset for selecting *service*
58 *endpoints*: descriptors of concrete URIs at which network services are available for the target
59 resource. An XRI resolver may optionally use the path and/or query components of an XRI to
60 select the service endpoint(s) to return to a consuming application.

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

62 • *HTTP.* As a resolution protocol, HTTP not only makes it easy to deploy XRI resolution
63 services (including proxy resolution services), but also allows them to employ both HTTP
64 security standards (e.g., HTTPS) and XML-based security standards (e.g., SAML). Although
65 less efficient than UDP, HTTP(S) is suitable for the higher level of abstraction represented by
66 XRIs and can take advantage of the full caching capabilities of modern web infrastructure.

67 • *XRDS documents.* This simple, extensible XML resource description format makes it easy to
68 describe the capabilities of any XRI-, IRI-, or URI-identified resource in a manner that can be
69 consumed by any XML-aware application (or even by non-XRI aware browsers via the use of
70 a proxy resolver).

71 • *Synonyms.* DNS uses the CNAME attribute to establish equivalence between domain names.
72 XRDS architecture includes four synonym elements (LocalID, EquivID, CanonicalID, and
73 CanonicalEquivID) to provide robust support for mapping XRIs, IRIs, or URIs to other XRIs,
74 IRIs, or URIs that identify the same target resource. This is particularly useful for discovering
75 and mapping persistent identifiers that are often required by trust infrastructures.

76 • *Redirects and Refs.* XRDS architecture includes two elements for distributed XRDS
77 document management: the Redirect element allows the same identifier authority to manage
78 multiple XRDS documents describing the same resource from different network locations,
79 and the Ref element allows one identifier authority to delegate all or part of an XRDS
80 document to another identifier authority.

81 • *Service endpoint descriptors.* DNS can use NAPTR records to do string transformations into
82 URIs representing network endpoints. XRDS documents have *service endpoint descriptors*—
83 elements that describe the set of URIs at which a particular type of service is available. Each
84 service endpoint may present a different type of data or metadata representing or describing
85 the identified resource. Thus XRI resolution can serve as a lightweight, interoperable
86 discovery mechanism for resource attributes available via HTTP(S), LDAP, UDDI, SAML,
87 WS-Trust, or other directory or discovery protocols.

88 1.2 Structure of this Specification

89 This specification is structured into the following major sections:

90 • *Conformance* (section 2) specifies the conformance targets and conformance claims for this
91 specification.

92 • *Namespaces* (section 3) specifies the XRI and XML namespaces and media types used for
93 the XRI resolution protocol.

94 • *XRDS Documents* (section 4) specifies a simple, flexible XML-based container for XRI
95 resolution metadata and/or other metadata describing a resource.

96 • *XRDS Synonyms* (section 5) specifies usage of the four XRDS synonym elements.

- 97 • *Discovering an XRDS Document from an HTTP(S) URI* (section 6) specifies how to obtain
98 XRDS metadata describing a resource, including synonyms for that resource, starting from
99 an HTTP(S) URI identifying the resource.
- 100 • *Inputs and Outputs* (section 7) specifies the standard input parameters and output formats for
101 XRI resolution.
- 102 • *Generic Authority Resolution* (section 8) specifies a simple resolution protocol for the
103 authority segment of an XRI using HTTP/HTTPS as a transport.
- 104 • *Trusted Authority Resolution* (section 9) specifies three extensions to generic authority
105 resolution for creating a chain of trust between the participating identifier authorities using
106 HTTPS connections, SAML assertions, or both.
- 107 • *Proxy Resolution* (section 10) specifies an HTTP(S) interface for an XRI resolver plus a
108 format for expressing an XRI as an HTTP(S) URI to provide backwards compatibility with
109 existing HTTP(S) infrastructure.
- 110 • *Redirect and Ref Processing* (section 11) specifies how a resolver follows a reference from
111 one XRDS document to another to enable federation of XRDS documents across multiple
112 network locations (Redirects) or identifier authorities (Refs).
- 113 • *Service Endpoint Selection* (section 12) specifies an optional second phase of resolution for
114 selecting a set of service endpoints from an XRDS document.
- 115 • *Synonym Verification* (section 13) specifies how a resolver can verify that one XRI, IRI, or
116 URI is an authorized synonym for another.
- 117 • *Status Codes and Error Processing* (section 14) specifies status reporting and error handling.
- 118 • *Use of HTTP(S)* (section 15) specifies how the XRI resolution protocol leverages features of
119 the HTTP(S) protocol.
- 120 • *Extensibility and Versioning* (section 16) describes how the XRI resolution protocol can be
121 easily extended and how new versions will be identified and accommodated.
- 122 • *Security and Data Protection* (section 17) summarizes key security and privacy
123 considerations for XRI resolution infrastructure.

124 1.3 Examples of XRI Resolution Requests and Responses

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

128 1.4 Terminology and Notation

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

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

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

137 Formatting conventions used in this document:

138 `Examples look like this.`

139 `| ABNF productions look like this.`

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

1.5 Normative References

- 142 **[DNSSEC]** D. Eastlake, *Domain Name System Security Extensions*,
 143 <http://www.ietf.org/rfc/rfc2535>, IETF RFC 2535, March 1999.
- 144 **[RFC2045]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
 145 *Part One: Format of Internet Message Bodies*,
 146 <http://www.ietf.org/rfc/rfc2045.txt>, IETF RFC 2045, November 1996.
- 147 **[RFC2046]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
 148 *Part Two: Media Types*, <http://www.ietf.org/rfc/rfc2046.txt>, IETF RFC
 149 2046, November 1996.
- 150 **[RFC2119]** S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*,
 151 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 152 **[RFC2141]** R. Moats, *URN Syntax*, <http://www.ietf.org/rfc/rfc2141.txt>, IETF RFC
 153 2141, May 1997.
- 154 **[RFC2483]** M. Mealling, R. Daniel Jr., *URI Resolution Services Necessary for URN*
 155 *Resolution*, <http://www.ietf.org/rfc/rfc2483.txt>, IETF RFC 2483, January
 156 1999.
- 157 **[RFC2616]** R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.
 158 Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*,
 159 <http://www.ietf.org/rfc/rfc2616.txt>, IETF RFC 2616, June 1999.
- 160 **[RFC2818]** E. Rescorla, *HTTP over TLS*, <http://www.ietf.org/rfc/rfc2818.txt>, IETF
 161 RFC 2818, May 2000.
- 162 **[RFC3023]** M. Murata, S. St-Laurent, D. Kohn, *XML Media Types*,
 163 <http://www.ietf.org/rfc/rfc3023.txt>, IETF RFC 3023, January 2001.
- 164 **[RFC3986]** T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier*
 165 *(URI): Generic Syntax*, <http://www.ietf.org/rfc/rfc3986.txt>, IETF RFC
 166 3986, January 2005.
- 167 **[RFC4234]** D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications:*
 168 *ABNF*, <http://www.ietf.org/rfc/rfc4234.txt>, IETF RFC 4234, October 2005.
- 169 **[RFC4288]** N. Freed, J. Klensin, *Media Type Specifications and Registration*
 170 *Procedures*, <http://www.ietf.org/rfc/rfc4288.txt>, IETF RFC 4288,
 171 December 2005.
- 172 **[SAML]** S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for*
 173 *the OASIS Security Assertion Markup Language (SAML) V2.0*,
 174 <http://www.oasis-open.org/committees/security>, March 2005.
- 175 **[Unicode]** The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined
 176 by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley,
 177 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1
 178 (<http://www.unicode.org/versions/Unicode4.0.1>) and by Unicode 4.1.0
 179 (<http://www.unicode.org/versions/Unicode4.1.0>), March, 2005.
- 180 **[UUID]** Open Systems Interconnection – *Remote Procedure Call*, ISO/IEC
 181 11578:1996, <http://www.iso.org/>, August 2001.
- 182 **[XML]** T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau,
 183 *Extensible Markup Language (XML) 1.0, Third Edition*, World Wide Web
 184 Consortium, <http://www.w3.org/TR/REC-xml/>, February 2004.
- 185 **[XMLDSig]** D. Eastlake, J. Reagle, D. Solo et al., *XML-Signature Syntax and*
 186 *Processing*, World Wide Web Consortium,
 187 <http://www.w3.org/TR/xmlsig-core/>, February, 2002.
- 188 **[XMLID]** J. Marsh, D. Veillard, N. Walsh, *xml:id Version 1.0*, World Wide Web
 189 Consortium, <http://www.w3.org/TR/2005/REC-xml-id-20050909>,
 190 September 2005.

191 **[XMLSchema]** H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, *XML Schema Part*
192 *1: Structures Second Edition*, World Wide Web Consortium,
193 <http://www.w3.org/TR/xmlschema-1/>, October 2004.

194 **[XMLSchema2]** P. Biron, A. Malhotra, *XML Schema Part 2: Datatypes Second Edition*,
195 World Wide Web Consortium, <http://www.w3.org/TR/xmlschema-2/>,
196 October 2004.

197 **[XRIMetadata]** D. Reed, *Extensible Resource Identifier (XRI) Metadata V2.0*,
198 <http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf>,
199 March 2005.

200 **[XRISyntax]** D. Reed, D. McAlpin, *Extensible Resource Identifier (XRI) Syntax V2.0*,
201 <http://docs.oasis-open.org/xri/xri/V2.0/xri-syntax-V2.0-cd-01.pdf>, March
202 2005.

203 1.6 Non-Normative References

204 **[XRIFAQ]** OASIS XRI Technical Committee, *XRI 2.0 FAQ*, [http://www.oasis-](http://www.oasis-open.org/committees/xri/faq.php)
205 [open.org/committees/xri/faq.php](http://www.oasis-open.org/committees/xri/faq.php), Work-In-Progress, March 2006.

206 **[XRIGuide]** [TODO]

207 **[XRIReqs]** G. Wachob, D. Reed, M. Le Maitre, D. McAlpin, D. McPherson,
208 *Extensible Resource Identifier (XRI) Requirements and Glossary v1.0*,
209 [http://www.oasis-](http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc)
210 [open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-](http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc)
211 [and-glossary-v1.0.doc](http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc), June 2003.

212 **[WikipediaXRI]** Wikipedia entry on XRI (Extensible Resource Identifier),
213 <http://en.wikipedia.org/wiki/XRI>, Wikipedia Foundation.

214 **[Yadis]** [TODO]

215 **2 Conformance**

216 This section specifies conformance targets and conformance claims.

217 **2.1 Conformance Targets**

218 The targets of this specification are:

- 219 1. XRDS clients, which provided a limited subset of the functionality of XRI resolvers.
- 220 2. XRDS servers, which provided a limited subset of the functionality of XRI authority
221 servers.
- 222 3. XRI resolvers, which may be either local resolvers or proxy resolvers.
- 223 4. XRI authority servers.

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

226 **2.2 XRDS Clients**

227 [TODO]

228 **2.3 XRDS Servers**

229 [TODO]

230 **2.4 XRI Resolvers**

231 **2.4.1 Local Resolvers**

232 [TODO]

233 **2.4.2 Proxy Resolvers**

234 [TODO]

235 **2.5 XRI Authority Servers**

236 [TODO]

237

3 Namespaces

238

3.1 XRI Namespaces for XRI Resolution

239

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.

240

241

242

243

3.1.1 XRIs Reserved for XRI Resolution

244

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

245

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 (versioned)	3.2
xri://\$xrd*(\$v*2.0)	Version 2.0 of above (using an XRI version identifier as defined in [XRIMetadata])	3.2

246

Table 2: XRIs reserved for XRI resolution.

247

3.1.2 XRIs Assigned to XRI Resolution Service Types

248

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

249

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

250

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.

251

252

253

3.2 XML Namespaces for XRI Resolution

254

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.

255

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

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

257 3.3 Media Types for XRI Resolution

258 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as
 259 defined by [RFC2046], particularly in HTTP Accept headers as specified in [RFC2616]. Table 5
 260 specifies the media types used for XRI resolution. Note that in XRI authority resolution, these
 261 media types MUST be passed as HTTP Accept header values. By contrast, in XRI proxy resolution
 262 these media types MUST be passed as query parameters in an HTTP(S) URI as specified in
 263 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]

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

265 To provide full control of XRI resolution, the media types specified in Table 5 accept the media
 266 type parameters defined in Table 6. Note that when these media type parameters are appended
 267 to a media type in the XRI proxy resolver interface, the semicolon character used to concatenate
 268 them MUST be percent-encoded as specified in section 10.3.

Parameter	Values	Usage	See Section
https	true or 1 false or 0	Specifies use of HTTPS trusted resolution	9.1
saml	true or 1 false or 0	Specifies use of SAML trusted resolution	9.2
sep	true or 1 false or 0	Specifies whether service endpoint selection should be performed	11
nodefault_t	true or 1 false or 0	Specifies whether a default match on a Type service endpoint selection element is allowed	12.3

nodefault_p	true or 1 false or 0	Specifies whether a default match on a Path service endpoint selection element is allowed	12.3
nodefault_m	true or 1 false or 0	Specifies whether a default match on a MediaType service endpoint selection element is allowed	12.3
uric	true or 1 false or 0	Specifies whether a resolver should automatically construct service endpoint URIs	12.7.1
refs	true or 1 false or 0	Specifies whether references should be followed during resolution (by default they are followed)	13
cid	true or 1 false or 0	Specifies whether automatic canonical ID verification should performed (by default it is performed)	13

Comment [DSR9]: New in ED06

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

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

271

4 XRDS Documents

272
273
274
275
276

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.

277

4.1 XRDS and XRD Namespaces

278
279
280
281
282
283
284
285
286
287

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.

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].

288
289
290
291

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.

292

4.2 XRD Elements and Attributes

293
294

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

295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320

```
<XRDS xmlns="xri://$xrds" ref="xri://(http%3A%2F%2Fexample.org)*foo">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*foo</Query>
    <Status code="100"/>
    <Expires>2005-05-30T09:30:10Z</Expires>
    <ProviderID>
      xri://(http%3A%2F%2Fexample.org)!1234
    </ProviderID>
    <LocalID>*baz</LocalID>
    <EquivID>
      xri://(http%3A%2F%2Fexample.org)!1234!5678
    </EquivID>
    <CanonicalID>
      xri://(https%3A%2F%2Fexample.org)!1234!5678
    </CanonicalID>
    <CanonicalEquivID>
      xri://=!4a76!c2f7!9033.78bd
    </CanonicalEquivID>
    <Service>
      <ProviderID>
        xri://(https%3A%2F%2Fexample.org)!1234
      </ProviderID>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <MediaType>application/xrds+xml</MediaType>
      <URI priority="10">http://resolve.example.com</URI>
      <URI priority="15">http://resolve2.example.com</URI>
```

```

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

```

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

358 4.2.1 Management Elements

359 The first set of elements are used to manage XRDs, particularly from the perspective of caching
360 and error handling. Note that due to conflicting processing rules, the XRD schema permits a
361 choice of either `xrd:XRD/xrd:Redirect` elements or `xrd:XRD/xrd:Ref` elements but not
362 both.

363 **xrd:XRD**

364 Container element for all other XRD elements. Includes an OPTIONAL `xml:id` attribute
365 of type `xs:ID`. This attribute is REQUIRED in trusted resolution to uniquely identify this
366 element within the containing `xrd:XRDS` document. It also includes an OPTIONAL
367 `xrd:idref` attribute of type `xs:idref`. This attribute is REQUIRED in trusted resolution
368 when an XRD element in a nested `xrd:XRDS` document must reference a previously
369 included XRD instance. See sections 4.3 and 11.1. Lastly, it includes a `version`
370 attribute that is OPTIONAL for uses outside of XRI resolution but REQUIRED for XRI
371 resolution as defined in section 4.3.2

372 **xrd:XRD/xrd:Query**

373 0 or 1 per `xrd:XRD` element. Expresses the XRI, IRI, or URI reference in URI-normal
374 form whose resolution results in this `xrd:XRD` element. See section 5.1.

375 **xrd:XRD/xrd>Status**

376 0 or 1 per `xrd:XRD` element. The contents of the element are a human-readable
377 message string describing the status of the response as determined by the resolver. It is
378 REQUIRED if the resolver must report certain error conditions. Contains a REQUIRED
379 attribute `code` of type `xs:int` that provides a numeric status code. Contains OPTIONAL
380 enumerated attributes `cid` and `ceid` that reports the results of CanonicalID verification
381 as defined in section 13.2.4. For XRI resolution, values of the Status element and `code`
382 attribute are defined in section 14.

Comment [DSR10]: Rewritten in ED06.

383 **xrd:XRD:xrdServerStatus**

384 0 or 1 per `xrd:XRD` element. Identical to `xrd:XRD/xrd>Status` except this element
385 reflects the status reported to an XRI resolver by an XRI authority server. See section
386 14.1.

Comment [DSR11]: New in ED06.

387 **xrd:XRD/xrd:Expires**

388 0 or 1 per `xrd:XRD` element. The date/time, in the form of `xs:dateTime`, after which
389 this XRD cannot be relied upon. To promote interoperability, this date/time value
390 SHOULD use the UTC "Z" time zone and SHOULD NOT use fractional seconds. A
391 resolver using this XRD MUST NOT use the XRD after the time stated here. A resolver
392 MAY discard this XRD before the time indicated in this result. If the HTTP transport
393 caching semantics specify an expiry time earlier than the time expressed in this attribute,
394 then a resolver MUST NOT use this XRD after the expiry time declared in the HTTP
395 headers per section 13.2 of [RFC2616]. See section 15.2.1.

396 **xrd:XRD/xrd:Redirect**

397 0 or more per `xrd:XRD` element. Choice between this or the `xrd:XRD/xrd:Ref`
398 element below. Type `xs:anyURI`. MUST be an absolute HTTP(S) URI. MUST be
399 processed by a resolver to locate another XRDS document authorized to describe the
400 target resource as defined in section 11.

Comment [DSR12]: New in ED06.

401 **xrd:XRD/xrd:Ref**

402 0 or more more per `xrd:XRD` element. Choice between this or the
403 `xrd:XRD/xrd:Redirect` element above. Type `xs:anyURI`. MUST be an absolute
404 XRI. MUST be processed by a resolver (depending on the value of the `refs` media type

405 parameter) to locate another XRDS document authorized to describe the target resource
 406 as defined in section 11.

407 4.2.2 Trust Elements

408 The second set of elements are for applications where trust must be established in the authority
 409 providing the XRD. These elements are OPTIONAL for generic authority resolution (section 8),
 410 but may be REQUIRED for specific types of trusted authority resolution (section 9) and
 411 CanonicalID verification (section 13.2).

412 **xrd:XRD/xrd:ProviderID**

413 0 or 1 per `xrd:XRD`. A unique identifier of type `xs:anyURI` for the parent authority
 414 providing this XRD. The value of this element **MUST** be a persistent identifier. There
 415 **MUST** be negligible probability that the value of this element will be assigned as an
 416 identifier to any other authority. For purposes of CanonicalID verification (section 13.2), it
 417 is **RECOMMENDED** to use a fully persistent XRI as defined in **[XRISyntax]**. If a URN
 418 **[RFC2141]** or other persistent identifier is used, it is **RECOMMENDED** to express it as an
 419 XRI cross-reference as defined in **[XRISyntax]**. Note that for XRI authority resolution, the
 420 authority identified by this element is the *parent* authority (the provider of the current
 421 XRD), not the *child* authority (the target of the current XRD). The latter is identified by the
 422 `xrd:XRD/xrd:Service/xrd:ProviderID` element inside a resolution service
 423 endpoint (see below).

424 **xrd:XRD/saml:Assertion**

425 0 or 1 per `xrd:XRD`. A SAML assertion from the parent authority (the provider of the
 426 current XRD) that asserts that the information contained in the current XRD is
 427 authoritative. Because the assertion is digitally signed and the digital signature
 428 encompasses the containing `xrd:XRD` element, it also provides a mechanism for the
 429 recipient to detect unauthorized changes since the last time the XRD was published.

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

435 4.2.3 Synonym Elements

436 In XRDS architecture, an identifier is a *synonym* of the query identifier (the identifier resolved to
 437 obtain the XRDS document) if it is not character-for-character equivalent but identifies the same
 438 target resource (the resource to which the identifier was assigned by the identifier authority). The
 439 normative rules for synonym usage in XRI resolution are specified in section 5.

440 **xrd:XRD/xrd:LocalID**

441 0 or more per `xrd:XRD` element. Type `xs:anyURI`. Accepts the optional global
 442 `xrd:priority` attribute (see section 4.3.3). Asserts a interchangeable synonym for the
 443 value of the `xrd:Query` element. **MUST** be assigned by the same parent authority
 444 providing the current XRD. Does not require verification. See section 5.2.1.

445 **xrd:XRD/xrd:EquiVID**

446 0 or more per `xrd:XRD` element. Type `xs:anyURI`. Accepts the optional global
 447 `priority` attribute (see section 4.3.3). Asserts any absolute synonym for the query
 448 identifier except a CanonicalID or CanonicalEquiVID (below). **MUST** be an absolute
 449 identifier. **MAY** be verified as defined in section 13.1. **MAY** be used for CanonicalEquiVID
 450 verification as defined in section 13.2.3. See section 5.2.2.

451 **xrd:XRD/xrd:CanonicalID**

452 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical synonym for the
453 query identifier which the parent authority is authoritative. MUST be an absolute identifier.
454 SHOULD be verified as defined in section 13.2. See section 5.2.3.

455 **`xrd:XRD/xrd:CanonicalEquiVid`**

456 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical synonym for the
457 query identifier regardless of identifier authority. MUST be an absolute identifier.
458 SHOULD be verified as defined in section 13.2.3. See section 5.2.4.

459 **4.2.4 Service Endpoint Descriptor Elements**

460 The next set of elements are used to describe service endpoints—the set of network endpoints
461 advertised in an XRD for performing further resolution, obtaining further metadata, or interacting
462 directly with the described resource. Again, because there can be more than one instance of a
463 service endpoint that satisfies a service endpoint selection query, or more than one instance of
464 these elements inside a service descriptor, these elements all accept the `global:priority`
465 attribute (see section 4.3.3). Note that due to conflicting processing rules, the XRD schema
466 permits allows use of only one of the following three elements with a service endpoint: `xrd:URI`,
467 `xrd:Redirect`, or `xrd:Ref`.

468 **`xrd:XRD/xrd:Service`**

469 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.
470 Referred to by the abbreviation *SEP*.

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

472 0 or more per `xrd:XRD/xrd:Service` element. Identical to the
473 `xrd:XRD/xrd:LocalID` element defined above except this synonym is assigned by the
474 provider of the service and not the parent authority for the XRD. MAY be used to provide
475 one or more identifiers by which the target resource SHOULD be identified in the context
476 of the service endpoint.

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

478 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the
479 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
480 elements. Type `xs:anyURI`. If present, it indicates a transport-level URI for accessing
481 the capability described by the parent `Service` element. For the service types defined for
482 XRI resolution in section 3.1.2, this URI MUST be an HTTP or HTTPS URI. Other
483 services may use other transport protocols. This element includes an optional `append`
484 attribute that governs construction of the final service endpoint URI as defined in section
485 12.7.

486 **`xrd:XRD/xrd:Service/xrd:Redirect`**

487 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the
488 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Ref` elements.
489 Type `xs:anyURI`. Identical to the `xrd:XRD/xrd:Redirect` element defined above
490 except processed only in the context of service endpoint selection. See section 11. [

Comment [DSR14]: New in ED06.

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

492 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the
493 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Redirect`
494 elements. Identical to the `xrd:XRD/xrd:Ref` element defined above except processed
495 only in the context of service endpoint selection. See section 11. [

Comment [DSR15]: Revised in ED06.

496 4.2.5 Service Endpoint Trust Elements

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

500 **xrd:XRD/xrd:Service/xrd:ProviderID**

501 0 or 1 per `xrd:XRD/xrd:Service` element. Identical to the
502 `xrd:XRD/xrd:ProviderID` above, except this identifies the provider of the *described*
503 *service endpoint* instead of the provider of the current XRD. In XRI resolution, this means
504 it identifies the *child authority* who will perform resolution of subsequent XRI
505 subsegments. In SAML trusted resolution, when a resolution request is made to the child
506 authority at this service endpoint, the contents of the `xrd:XRD/xrd:ProviderID`
507 element in the response MUST match the content of this element for correlation as
508 defined in section 9.2.5. The same usage MAY apply to other services not defined in this
509 specification. Authors of other specifications employing XRD service endpoints SHOULD
510 define the scope and usage of this element, particularly for trust verification.

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

512 0 or 1 per `xrd:XRD/xrd:Service` element. This element provides the digital signature
513 metadata necessary to validate interaction with the resource identified by the
514 `xrd:XRD/xrd:Service/xrd:ProviderID` (above). In XRI resolution, this element
515 comprises the key distribution method for SAML trusted authority resolution as defined in
516 section 9.2.5. The same usage MAY apply to other services not defined in this
517 specification.

518 4.2.6 Service Endpoint Selection Elements

519 The final set of service endpoint descriptor elements are used in XRI resolution to select service
520 endpoints. They include two global attributes used for this purpose: `match` and `select`. See
521 sections 12.3.2 and 12.4.2.

522 **xrd:XRD/xrd:Service/xrd:Type**

523 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI`
524 that identifies the type of capability available at this service endpoint. See section 3.1.2
525 for the resolution service types defined in this specification. If a service endpoint does not
526 include at least one `xrd:Type` element, the service type is effectively described by the
527 type of URI specified in the `xrd:XRD/xrd:Service/xrd:URI` element, i.e., an HTTP
528 URI specifies an HTTP service. See section 12.3.6 for Type element matching rules.

529 **xrd:XRD/xrd:Service/xrd:Path**

530 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string
531 value meeting the `xri-path` production defined in section 2.2.3 of [XRISyntax]. See
532 section 12.3.7 for Path element matching rules.

533 **xrd:XRD/xrd:Service/xrd:MediaType**

534 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of
535 content available at this service endpoint. The value of this element MUST be of the form
536 of a media type defined in [RFC2046]. See section 3.3 for the media types used in XRI
537 resolution. See section 12.3.8 for MediaType element matching rules.

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

542 4.3 XRD Attribute Processing Rules

543 4.3.1 ID Attribute

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

555 Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their
556 XML document order MUST match the order in which they were resolved (see section 8.1.2).
557 Also, if reference processing requires the same XRD to be included in an XRDS document twice
558 (via a nested XRDS document), that XRD MUST reference the previous instance using the
559 `xrd:XRD/@xml:idref` attribute as defined in section 11.

560 4.3.2 Version Attribute

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

565 General rules about versioning of the XRI resolution protocol are defined in section 16.2. Specific
566 for processing of the XRD version attribute are specified in section 16.2.4.

567 4.3.3 Priority Attribute

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

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

- 576 1. The consuming application SHOULD select the element instance with the lowest numeric
577 value of the priority attribute. For example, an element with priority attribute value of “10”
578 should be selected before an element with a priority attribute value of “11”, and an
579 element with priority attribute value of “11” should be selected before an element with a
580 priority attribute value of “25”. Zero is the highest priority attribute value. Null is the lowest
581 priority attribute value—it is the equivalent of a value of infinity. It is RECOMMENDED to
582 use a large finite value (100 or more) rather than a null value.
- 583 2. If an element has no priority attribute, its priority attribute value is considered to be null,
584 i.e., the lowest possible priority value. Rather than omitting a priority attribute, it is
585 RECOMMENDED that XRI authorities follow the standard practice in DNS and set the
586 default priority attribute value to “10”.
- 587 3. If two or more instances of the same element type have identical priority attribute values
588 (including the null value), the consuming application SHOULD select one of the instances

589 at random. This consuming application SHOULD NOT simply choose the first instance
590 that appears in XML document order. *This is important in order to support intentional load*
591 *balancing semantics.*

592 4. An element selected according to these rules is referred to in this specification as *the*
593 *highest priority element*. If this element is subsequently disqualified from the set of
594 qualified elements, the next element selected according to these rules is referred to as
595 *the next highest priority element*. If an XRI resolution operation specifying selection of the
596 highest priority element fails, the resolver SHOULD attempt to select the next highest
597 priority element unless otherwise specified. This process SHOULD be continued for all
598 other instances of the qualified elements until success is achieved or all instances are
599 exhausted.

600 4.4 XRI and IRI Encoding Requirements

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

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

609 Note: XRIs composed entirely of valid URI characters and which do not use XRI parenthetical
610 cross-reference syntax do not require escaping in the transformation to URI-normal form.
611 However XRIs that use characters valid only in IRIs or that use XRI parenthetical cross-reference
612 syntax may require percent encoding in the transformation to URI-normal form as explained in
613 section 2.3 of [XRISyntax].

614 5 XRDS Synonyms

615 XRDS architecture includes support for *synonyms*—XRIs, IRIs, or URIs that are not character-for-
616 character equivalent, but which identify the same target resource (in the same context, or across
617 different contexts). Table 7 lists the four synonym elements supported in XRDS.

XRD Synonym Element	Cardinality	Resolution Scope	Assigning Authority	Resolves to different XRD?
LocalID	Zero-or-more	Local	MUST be the parent authority	MUST NOT
EquivID	Zero-or-more	Global	Any authority	SHOULD
CanonicalID	Zero-or-one	Global	MUST be the parent authority	MUST NOT
CanonicalEquivID	Zero-or-one	Global	Any authority	SHOULD

618 Table 7: The four XRDS synonym elements.

619 This section specifies the normative rules for usage of each XRDS synonym element.

620 5.1 Query Identifiers

621 The identifier that is resolved to obtain an XRDS document is called the *fully-qualified query*
622 *identifier*. A fully-qualified query identifier may be either:

- 623 1. A valid absolute HTTP(S) URI that does not contain an XRI.
- 624 2. A valid absolute XRI, either in a standard XRI form as defined in [XRISyntax], or
625 encoded in an HTTP(S) URI (called an *HXRI*) as specified in section 10.2.

626 5.1.1 HTTP(S) URI Query Identifiers

627 If the fully-qualified query identifier is an absolute HTTP(S) URI, the XRDS document to which it
628 resolves (via the protocol specified in section 6) MUST contain a single XRD. This XRD MAY
629 include an `xrd:Query` element; if present, the value MUST be equivalent to the original HTTP(S)
630 URI query identifier.

631 In this single XRD, all synonym elements in Table 7 assert synonyms for the original HTTP(S)
632 URI.

633 5.1.2 XRI Query Identifiers

634 If the fully-qualified query identifier is an absolute XRI, the XRDS document to which it resolves
635 (via the protocol specified in section 8.1.2) MAY contain multiple XRDs, each XRD corresponding
636 to one subsegment of the authority segment of the XRI. Each XRD SHOULD include an
637 `xrd:Query` element that echos back the XRI subsegment described by this XRD. This is called
638 the *local query identifier*, because it represents just one subsegment of the fully-qualified query
639 identifier.

640 At any point in the XRI resolution chain, the combination of the community root authority XRI
641 (section 8.1.3) plus all local query identifiers resolved in all XRDs up to that point is called the
642 *current fully-qualified query identifier*. When the resolution chain is complete, the current fully-
643 qualified query identifier is equal to the starting fully-qualified query identifier.

644 In each XRD in the resolution chain, the LocalID element asserts a synonym for the local query
645 identifier, and the other three synonym elements in Table 7 (EquivID, CanonicalID, and
646 CanonicalEquivID) assert a synonym for the current fully-qualified query identifier.

647 5.2 XRD Synonym Elements

648 5.2.1 LocalID

649 In an XRD, a synonym for the local query identifier is asserted using the `xrd:LocalID` element.
650 LocalIDs may be used at both the authority level of an XRD (as a child of the root `xrd:XRD`
651 element) and the service level (as a child of the root `xrd:XRD/xrd:Service` element).

652 At the authority level, the value of the `xrd:XRD/xrd:LocalID` element asserts a synonym that
653 is interchangeable with the contents of the `xrd:Query` element in the XRD. This means an XRI
654 resolver MAY use it as an alternate key for that XRD in its cache. Resolution of a LocalID from
655 the same parent authority MUST return the same XRD as the XRD asserting the LocalID
656 synonym, i.e., an XRD containing the same elements and values (with the exception of the values
657 of the `xrd:Query`, `xrd:Expires`, and `xrd:XRD/xrd:LocalID` elements).

658 If the parent authority has assigned a persistent local identifier to the resource described by an
659 XRD, it SHOULD return this persistent identifier as an `xrd:XRD/xrd:LocalID` value in any
660 resolution response for a reassignable local identifier for the same resource. The reverse MAY
661 also be true, however parent authorities MAY adopt privacy or other policies that restrict the
662 reassignable synonyms returned for any particular resolution request.

663 At the service level, the `xrd:XRD/xrd:Service/xrd:LocalID` element MAY be used to
664 express either a local or global identifier for the target resource in the context of the specific
665 service being described. If present, consuming applications SHOULD use the value of the highest
666 priority instance of the `xrd:XRD/xrd:Service/xrd:LocalID` element to identify the target
667 resource in the context of this service endpoint. If not present, consuming applications SHOULD
668 choose a synonym as defined in section 5.5.

669 SPECIAL SECURITY CONSIDERATIONS: A parent authority SHOULD NOT permit a child
670 authority to edit a LocalID value in an XRD without authenticating the child authority and verifying
671 that the child authority is authorized to use this LocalID value either at the authority level and/or
672 the service level.

673 5.2.2 EquivID

674 In an XRD, any synonym for the current fully-qualified query identifier *except* a CanonicalID or a
675 CanonicalEquivID (see below) is asserted using the `xrd:EquivID` element. Unlike a LocalID, an
676 EquivID MAY be issued by any identifier authority; it is NOT REQUIRED to be issued by the
677 parent authority.

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

680 An EquivID element is OPTIONAL in an XRD except in two cases:

- 681 1. When it is REQUIRED as a backpointer to verify another EquivID element in a different
682 XRD as specified in section 13.1.
- 683 2. When it is REQUIRED as a backpointer to verify a CanonicalEquivID element as
684 specified in section 13.2.3.

685 SPECIAL SECURITY CONSIDERATIONS: An EquivID synonym SHOULD NOT be trusted
686 unless it is verified. This function is not performed automatically by XRI resolvers but may be
687 easily performed by consuming applications using one additional XRI resolution call as specified
688 in section 13.1. A parent authority SHOULD NOT permit a child authority to edit the EquivID value
689 in an XRD without authenticating the child authority and verifying that the child authority is

690 authorized to use this EquivID value. A parent authority MUST NOT assert an EquivID element if
691 the identifier authority to whom it points is not authorized to make a CanonicalEquivID assertion.

692 5.2.3 CanonicalID

693 The purpose of the `xrd:CanonicalID` element is to assert a canonical synonym for the current
694 fully-qualified query identifier for which the parent authority is authoritative. A CanonicalID MUST
695 meet all the requirements of an EquivID plus the following:

- 696 1. It MUST be an identifier for which the parent authority is the final authority. This means it
697 MUST resolve to the same XRD as the current fully-qualified query identifier, i.e., an XRD
698 containing the same elements and values (with the exception of the values of the
699 `xrd:Query`, `xrd:Expires`, and `xrd:XRD/xrd:LocalID` elements).
- 700 2. If it is any XRI except a community root authority XRI (section 8.1.3), it MUST be a direct
701 child of the parent authority's own CanonicalID. For example, if the CanonicalID asserted
702 for a target resource is `@!1!2!3`, then the CanonicalID for the parent authority issuing
703 the XRD must be `@!1!2`.
- 704 3. Once assigned, a parent authority SHOULD NEVER: a) change or reassign a
705 CanonicalID value, or b) stop asserting a CanonicalID element in an XRD in which it has
706 been asserted. For this reason, it is STRONGLY RECOMMENDED to use a persistent
707 identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

708 As a best practice, a parent authority SHOULD ALWAYS publish a CanonicalID element in an
709 XRD, even if the value is equivalent to the current fully-qualified query identifier. This practice:

- 710 • Makes it unambiguous to consuming applications which absolute synonym they should use to
711 identify the target resource in the context of the parent authority.
- 712 • Enables child authorities to issue their own verifiable CanonicalIDs.
- 713 • Enables verification of a CanonicalEquivID if asserted (below).

714 SPECIAL SECURITY CONSIDERATIONS: A CanonicalID synonym SHOULD NOT be trusted
715 unless it is verified. CanonicalID verification is performed automatically during resolution by an
716 XRI resolver unless this function is explicitly turned off; see section 13. A parent authority
717 SHOULD NOT permit a child authority to edit the CanonicalID value in an XRD without
718 authenticating the child authority and verifying that the child authority is authorized to use this
719 CanonicalID value.

720 5.2.4 CanonicalEquivID

721 The purpose of the `xrd:CanonicalEquivID` element is to assert a canonical synonym for the
722 fully-qualified query identifier for which the parent authority MAY NOT be authoritative. A
723 CanonicalEquivID MUST meet all the requirements of an EquivID plus the following:

- 724 1. In order for the value of the `xrd:CanonicalEquivID` element to be verified: a) the
725 XRD in which it appears MUST include a CanonicalID that can be verified as specified in
726 section 13.1, and b) the XRD to which it resolves MUST include an EquivID backpointer
727 to this CanonicalID as specified in section 13.2.3.
- 728 2. For the same reasons as with a CanonicalID, it is STRONGLY RECOMMENDED to use
729 a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

730 As a best practice, a parent authority SHOULD publish a CanonicalEquivID in an XRD if
731 consuming applications SHOULD be able to identify the target resource using a different identifier
732 than: a) the current fully-qualified query identifier, or b) the CanonicalID.

733 SPECIAL SECURITY CONSIDERATIONS: A CanonicalEquivID synonym SHOULD NOT be
734 trusted unless it is verified. Verification of the value of the CanonicalEquivID element in the final

735 XRD in an XRDS document is performed automatically during resolution by an XRI resolver
736 unless this function is explicitly turned off; see section 13. A parent authority SHOULD NOT
737 permit a child authority to edit the CanonicalEquiVID value in an XRD without authenticating the
738 child authority and verifying that the child authority is authorized to use this CanonicalEquiVID
739 value.

Comment [DSR16]: Rewritten in ED06.

740 5.3 Redirect and Ref Elements

741 While similar in some ways to a synonym element, the `xrd:Redirect` and `xrd:Ref` elements
742 MUST NOT be used to assert a synonym. Instead their purpose is to assert that a different XRDS
743 document is authorized to serve as an equally valid descriptor of the target resource. These
744 elements enable complete separation of the semantics of synonym assertions vs. distributed
745 XRDS document authorization assertions.

746 In the same way as a LocalID, both a Redirect and a Ref may be used at the authority level of an
747 XRD (as a child of the root `xrd:XRDS` element) and at the service level (as a child of the root
748 `xrd:XRDS/xrd:Service` element). The complete rules for Redirect and Ref processing in XRI
749 resolution are specified in section 11.

750 If two independent resources are later merged into the same resource, e.g., two businesses
751 merging into one, the use of an EquiVID, CanonicalID, or CanonicalEquiVID element SHOULD be
752 combined with the use of a Redirect or Ref element to provide the semantics of BOTH identifier
753 synonymity and authorized XRDS document equivalence.

754 SPECIAL SECURITY CONSIDERATIONS: A parent authority SHOULD NOT permit a child
755 authority to edit a Redirect or Ref value in an XRD without authenticating the child authority and
756 verifying that the child authority is authorized to use this Redirect or Ref value either at the
757 authority level and/or the service level.

758 5.4 Synonym Verification

759 For security purposes, it is STRONGLY RECOMMENDED that a consuming application not rely
760 on EquiVID, CanonicalID, or CanonicalEquiVID synonyms unless they are verified.

- 761 • EquiVID verification is not performed automatically by XRI resolvers but may be easily
762 performed by consuming applications using one additional XRI resolution call as specified in
763 section 13.1.
- 764 • CanonicalID and CanonicalEquiVID verification are performed automatically by XRI resolvers
765 (unless this function is explicitly turned off) as specified in section 13.2. Attributes of the
766 `xrd:XRDS/xrd:Status` element report whether the CanonicalID and CanonicalEquiVID were
767 present and verified as specified in section 13.2.4.

768 5.5 Synonym Selection

769 The policies applied by a consuming application to select a synonym to identify a target resource
770 are out of scope for this specification. However the following are RECOMMENDED best
771 practices:

- 772 • Only select a verified synonym (see above).
- 773 • Select a persistent synonym, particularly if a long term or immutable reference is required. If a
774 persistent synonym is present, other reassignable synonyms (including the current fully-
775 qualified query identifier) SHOULD be treated only as temporary identifiers.
- 776 • Select a CanonicalID if present, verified, and persistent. This identifier SHOULD be used
777 whenever referencing the target resource in the context of the parent authority issuing the
778 CanonicalID.
- 779 • If possible, *also* select a CanonicalEquiVID if present, verified, and persistent. This identifier
780 SHOULD be used as a reference to the target resource in any context.

- 781 • When selecting a synonym to use in the context of a specific service endpoint, follow the
782 recommendations for use of the `xrd:XRD/xrd:Service/xrd:LocalID` element as
783 specified in section 5.2.1.

784 **6 Discovering an XRDS Document from an**
785 **HTTP(S) URI**

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

790 **6.1 Overview**

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

- 792 1. *HEAD protocol*: using an HTTP(S) HEAD request to obtain a header with XRDS
793 document location information as specified in section 6.2.
- 794 2. *GET protocol*: using an HTTP(S) GET request with content negotiation as specified in
795 section 6.3.

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

799 **6.2 HEAD Protocol**

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

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

- 805 1. An `X-XRDS-Location` response-header.
- 806 2. A content type response-header specifying the content type `application/xrds+xml`.

807 If the response includes the first option above, the value of the `X-XRDS-Location` response-
808 header **MUST** be an HTTP(S) URI which gives the location of an XRDS document describing the
809 target resource. The XRDS client **MUST** then request this document as specified in section 6.3.

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

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

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

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

817 **6.3 GET Protocol**

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

821 The XRDS server response **MUST** be one of four options:

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

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

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

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

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

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

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

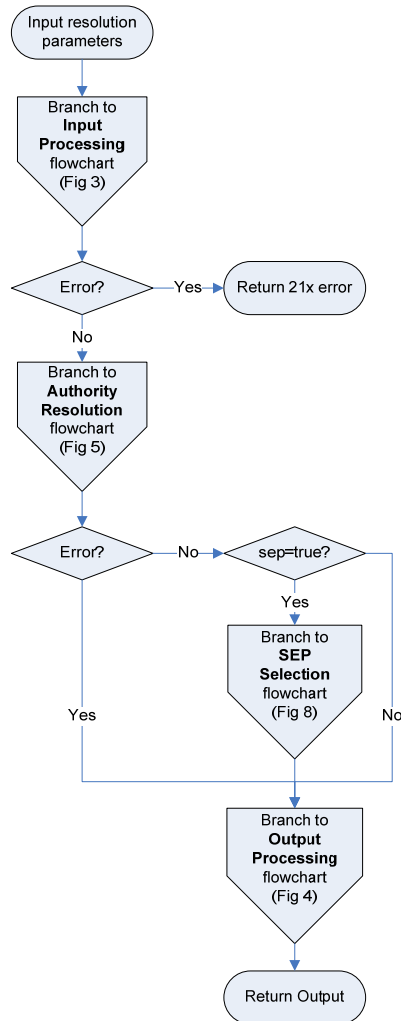
844 Note: If the XRDS server supports content negotiation, the response SHOULD include a vary:
845 header to allow caches to properly interpret future requests. This header SHOULD be present
846 even in the case where the HTML page is returned (instead of an XRDS document).

847

7 XRI Resolution Inputs and Outputs

848
849
850

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). A top-level flowchart of this function is shown in Figure 2.



851

852 *Figure 2: Top-level flowchart of XRI resolution phases.*

853
854
855
856

This section defines the logical inputs and outputs of this function together with their processing rules, 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.

857 **7.1 Inputs**

858 Table 8 summarizes the logical input parameters to XRI resolution and whether they are
 859 applicable in the authority resolution phase (sections 8 and 9) or the service endpoint selection
 860 phase (section 12). In this specification, references to these parameters use the logical names in
 861 the first column. Local APIs MAY use different names for these parameters and MAY define
 862 additional parameters.

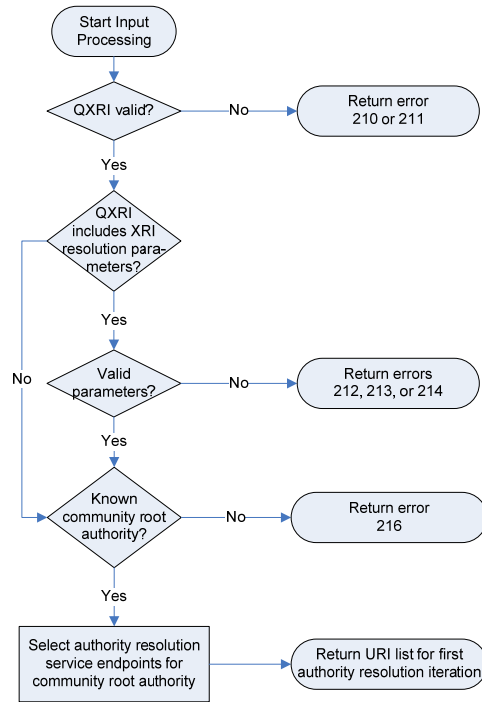
Logical Input Parameter Name	Type	Required/Optional	Default	Resolution Phase	Section
QXRI (query XRI) including Authority String, Path String, and Query String	xs:anyURI	Required	N/A	Authority resolution	7.1.1
Resolution Output Format	xs:string (media type)	Optional	Null	Authority resolution	7.1.2
Service 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

863 *Table 8: Input parameters for XRI resolution.*

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

- 866 1. The presence of an input parameter or an XRD element with an empty value is treated as
 867 equivalent to the absence of that input parameter or XRD element.
- 868 2. From a programmatic standpoint, both conditions above are considered as equivalent to
 869 setting the value of that parameter or element to null.
- 870 3. For all Boolean parameters: a) the string values `true` and `false` MUST be considered
 871 case-insensitive (lowercase is RECOMMENDED), b) the values `true` and `1` MUST be
 872 considered equivalent, b) the values `false` and `0` MUST be considered equivalent.

873 Figure 3 is a flowchart (non-normative) illustrating the processing of input parameters.



874

875 *Figure 3: Input processing flowchart.*

876 The following sections specify additional validation and usage requirements that apply to
877 particular input parameters.

878 **7.1.1 QXRI (Authority String, Path String, and Query String)**

879 The QXRI (query XRI) is the only REQUIRED input parameter. Per [XRISyntax], a QXRI consists
 880 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.

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

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

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

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

896 **7.1.2 Resolution Output Format**

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

- 898 • The media type for the resolution response.
- 899 • Whether generic or trusted resolution must be used by the resolver.
- 900 • Whether references should be followed during resolution.
- 901 • Whether CanonicalID verification should **not** be performed during resolution.
- 902 • Whether service endpoint selection should be performed on the final XRD.
- 903 • Whether default matches should be ignored during service endpoint selection.

- 904 • [Whether URIs should automatically be constructed in the final XRD.](#)

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

- 906 1. The value of Resolution Output Format MUST be one of the values specified in Table 5
907 and MAY include any of the media type parameters specified in Table 6.
- 908 2. If the value of the `https` media type parameter is `true`, the resolver MUST use the
909 HTTPS trusted authority resolution protocol specified in section 9.1 (or return an error
910 indicating this is not supported).
- 911 3. If the value of the `saml` media type parameter is `true`, the resolver MUST use the SAML
912 trusted authority resolution protocol specified in section 9.2 (or return an error indicating
913 this is not supported).
- 914 4. If the value of both the `https` and `saml` media type parameters are `true`, the resolver
915 MUST use the HTTPS+SAML trusted authority resolution protocol specified in section 9.3
916 (or return an error indicating this is not supported).
- 917 5. If the value of the `cid` media type parameter is `true` or null, or if the parameter is
918 absent, the resolver MUST perform CanonicalID verification as specified in section 13.2.
919 If the value of the `cid` media type parameter is `false`, the resolver MUST NOT perform
920 CanonicalID verification.
- 921 6. If the value of the `refs` media type parameter is `true` or null, or if the parameter is
922 absent, the resolver MUST perform Ref processing as specified in section 11. If the value
923 of the `refs` media type parameter is `false`, the resolver MUST NOT perform Ref
924 processing and must return an error if a Ref is encountered as specified in section 11.
- 925 7. If the value of the `sep` media type parameter is `true`, the resolver MUST perform service
926 endpoint selection on the final XRD (or return an error indicating this is not supported). If
927 the value of the `sep` media type parameter is `false` or null, or if the parameter is absent,
928 the resolver MUST NOT perform service endpoint selection on the final XRD unless it is
929 required to produce a URI List or HTTP(S) redirect. See section 7.2.
- 930 8. If the value of the `nodefault_r`, `nodefault_p`, or `nodefault_m` media media type
931 parameter is `true`, the resolver MUST ignore default matches on the corresponding
932 service endpoint selection element categories as specified in section 12.3.2.
- 933 9. If the value of the `uric` media type parameter is `true`, the resolver MUST perform
934 service endpoint URI construction as specified in section 12.7.1. If the value of the `uric`
935 media type parameter is `false` or null, or if the parameter is absent, the resolver MUST
936 NOT perform service endpoint URI construction.

Deleted: or null, or if the parameter is absent,

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

939 7.1.3 Service Type

940 The Service Type is an OPTIONAL value of type `xs:anyURI` used to request a specific type of
941 service in the service endpoint selection phase (section 10). The value of this parameter MUST
942 be a valid absolute XRI, IRI, or URI in URI-normal form as defined by **[XRISyntax]**. (Note that
943 URI-normal form is specified so this parameter may be passed to a proxy resolver in a QXRI
944 query parameter as defined in section 10.) The Service Type values defined for XRI resolution
945 services are specified in section 3.1.2. The Type element matching rules are specified in section
946 12.3.6.

947 7.1.4 Service Media Type

948 The Service Media Type is an OPTIONAL string used to request a specific media type in the
949 service endpoint selection phase (section 10). The value of this parameter MUST be a valid

950 media type as defined by **[RFC2046]**. The Service Media Type values defined for XRI resolution
951 services are specified in section 3.3. The MediaType element matching rules are specified in
952 section 12.3.8.

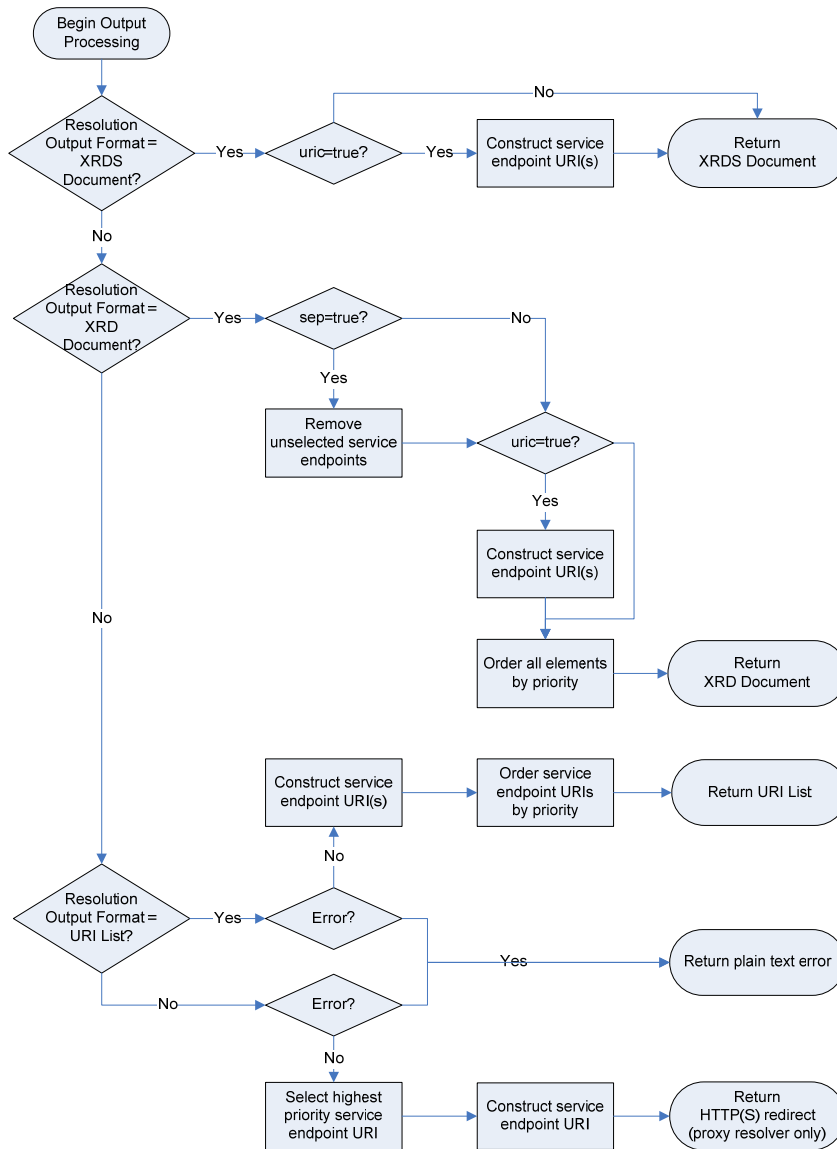
953 **7.2 Outputs**

954 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>

955 Table 10: Outputs of XRI resolution.

956 Figure 4 is a flowchart illustrating the process of producing these output formats once authority
 957 resolution and optional service endpoint selection is complete.



958
 959 *Figure 4: Output processing flowchart.*

960 The following sections provide additional construction and validation requirements.

961 7.2.1 XRDS Document

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

- 964 1. The output MUST be a valid XRDS document according to the schema defined in
965 Appendix A. In addition, any nested XRDS documents included as a result of Redirect
966 and Reference processing (section 13) must also be valid.
- 967 2. The XRDS document MUST contain an ordered list of `xrd:XRD` elements—one for each
968 authority subsegment successfully resolved by the resolver client. This list MUST appear
969 in the same order as the corresponding subsegments in the Authority String.
- 970 3. Each of the contained XRD elements must be a valid XRD document according to the
971 schema defined in Appendix A.
- 972 4. The XRD elements MUST conform to the additional requirements in section 4.
- 973 5. If the value of the `saml` parameter of the Resolution Output Format is `true`, the XRD
974 elements MUST conform to the additional requirements in section 9.2.
- 975 6. If Redirect or Ref processing is necessary during the authority resolution or service
976 endpoint selection process, it MUST result in a nested XRDS document as defined in
977 section 11.
- 978 7. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST
979 be performed as defined in section 12, even if the values of all three service endpoint
980 selection input parameters (Service Type, Path String, and Service Media Type) are null.
- 981 8. IMPORTANT: No filtering of the final XRD is performed when returning an XRDS
982 document. Filtering is only performed when the requested Resolution Output Format is
983 an XRD document – see the next section.
- 984 9. If the value of the `cid` media type parameter is `true`, synonym verification MUST be
985 reported using the `xrd:Status` element of each XRD in the XRDS document as
986 defined in section 13.
- 987 10. If the output is an error, this error MUST be returned using the `xrd:Status` element of
988 the final XRD in the XRDS document as defined in section 14.

989 7.2.2 XRD Document

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

- 992 1. The output MUST be a valid XRD document according to the schema defined in
993 Appendix A.
- 994 2. The XRD elements MUST conform to the additional requirements in section 4.
- 995 3. If the value of the `saml` parameter of the Resolution Output Format is `true`, the XRD
996 element MUST conform to the additional requirements in section 9.2.
- 997 4. If the value of the `sep` media type parameter is `false` or null, or if this parameter is
998 absent, the XRD MUST be the final XRD in the XRDS document produced as a result of
999 authority resolution. Service endpoint selection or any other filtering of the XRD
1000 document MUST NOT be performed.
- 1001 5. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST
1002 be performed as defined in section 12, even if the values of all three service endpoint
1003 selection input parameters (Service Type, Service Media Type, and Path String) are null.
- 1004 6. If service endpoint selection is performed, the only `xrd:Service` elements in the XRD
1005 document MUST be those selected according to the rules specified in section 12. If no
1006 service endpoints were selected by those rules, no `xrd:Service` elements will be

Formatted: 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

1007 present. In addition, all elements in the XRD document that are subject to the global
1008 `priority` attribute (even if the attribute is absent or null) MUST be returned in order of
1009 highest to lowest priority as defined in section 4.3.3. Any other filtering of the XRD
1010 document MUST NOT be performed. Note that this means that if the XRD document
1011 includes a SAML signature element as defined in section 9.2, this element is still returned
1012 in the XRD document even though it may not be able to be verified by a consuming
1013 application.

1014 7. If the value of the `cid` media type parameter is `true`, synonym verification MUST be
1015 reported using the `xrd:Status` element of each XRD in the XRDS document as
1016 defined in section 13.

1017 8. If the output is an error, this error MUST be returned using the `xrd:Status` element as
1018 defined in section 14.

1019 7.2.3 URI List

1020 If the value of the Resolution Output Format parameter is `text/uri-list`, the following rules
1021 apply.

- 1022 1. For this output, service endpoint selection is REQUIRED, even if the values of all three
1023 service endpoint selection input parameters (Service Type, Service Media Type, and
1024 Path String) are null.
- 1025 2. If authority resolution and service endpoint selection are both successful, the output
1026 MUST be a valid URI List as defined by section 5 of **[RFC2483]**.
- 1027 3. If, after applying the service endpoint selection rules, more than one service endpoint is
1028 selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as
1029 defined in section 4.3.3.
- 1030 4. If the final selected `xrd:XRD/xrd:Service` element contains a
1031 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
1032 element, Redirect and Ref processing MUST be performed as described in section 11.
- 1033 5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)
1034 MUST be constructed as defined in section 12.7.
- 1035 6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`
1036 elements within the selected `xrd:Service` element as defined in section 4.3.3. When
1037 two or more of the source `xrd:URI` elements have equal priority, their constructed URIs
1038 SHOULD be returned in random order. Any other filtering of the URI list MUST NOT be
1039 performed.
- 1040 7. If the output is an error, it MUST be returned with the content type `text/plain` as
1041 defined in section 14.

1042 7.2.4 HTTP(S) Redirect

1043 In XRI proxy resolution, the Resolution Output Format parameter may be null. In this case the
1044 output of a proxy resolver is an HTTP(S) redirect as defined in section 10.6.

1045 8 Generic Authority Resolution

1046 As discussed in section 1.1 and illustrated in Figure 2, authority resolution is the first phase of XRI
1047 resolution. This phase applies only to resolving the subsegments in the Authority String of the
1048 QXRI. The Authority String may identify either an *XRI authority* or an *IRI authority* as described in
1049 section 2.2.1 of [XRISyntax].

1050 XRI authorities and IRI authorities have different syntactic structures, partially due to the higher
1051 level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved
1052 to XRDS documents one subsegment at a time as specified in section 8.1. IRI authorities, since
1053 they are based on DNS names or IP addresses, are resolved into an XRDS document through a
1054 special HTTP(S) request using the entire IRI authority segment as specified in section 8.1.11.

1055 8.1 XRI Authority Resolution

1056 8.1.1 Service Type and Service Media Type

1057 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	OPTIONAL (see important note below)

1058 *Table 11: Service Type and Service Media Type values for generic authority resolution.*

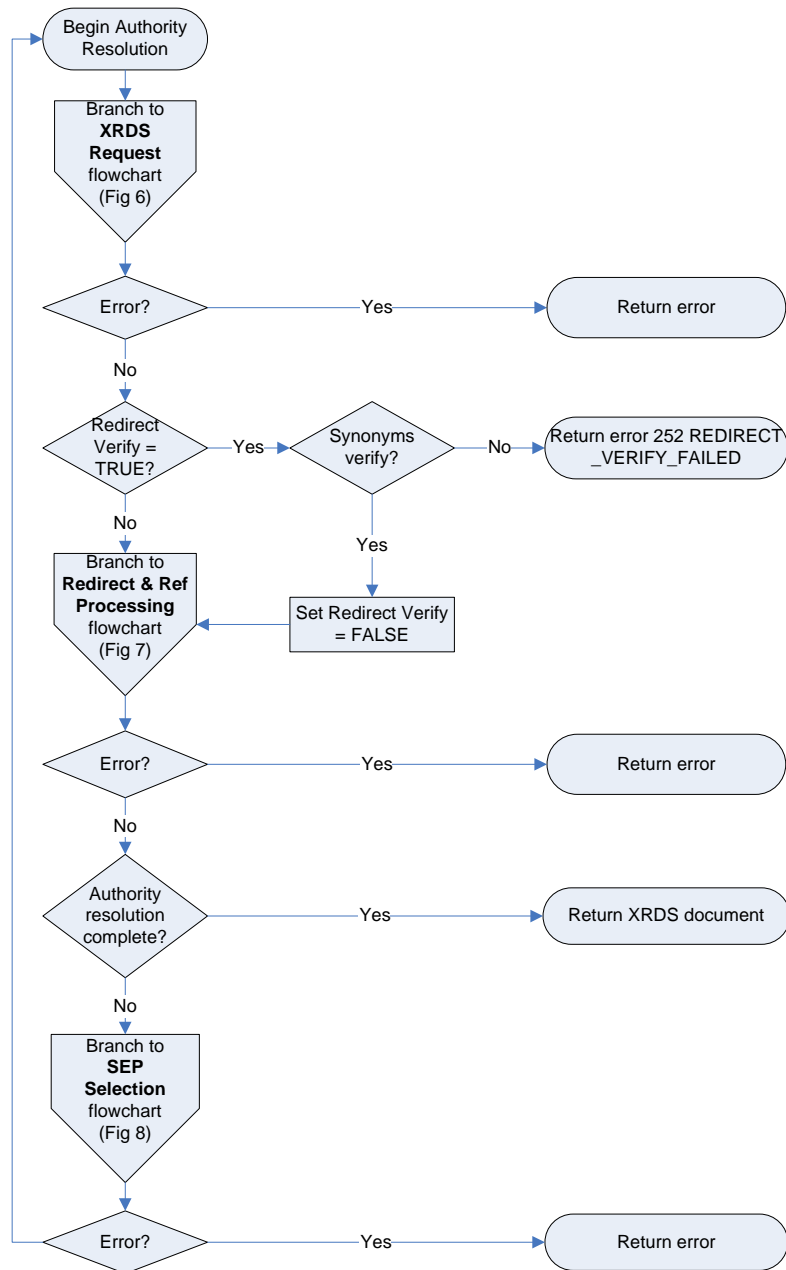
1059 A generic authority resolution service endpoint advertised in an XRDS document **MUST** use the
1060 Service Type identifier and **MAY** use the Service Media Type identifier defined in Table 11.

1061 **BACKWARDS COMPATABILITY NOTE:** Earlier drafts of this specification used a media type
1062 parameter called `trust`. This has been deprecated in favor of new parameters for each trusted
1063 resolution option, i.e., `https="true"` and `saml="true"`. However implementations **SHOULD**
1064 consider the following values equivalent both for the purpose of service endpoint selection within
1065 XRDS documents and as HTTP(S) Accept header values in XRI authority resolution requests:

```
1066 application/xrds+xml  
1067 application/xrds+xml;trust=none  
1068 application/xrds+xml;https=false  
1069 application/xrds+xml;saml=false  
1070 application/xrds+xml;https=false;saml=false  
1071 application/xrds+xml;saml=false;https=false
```

1072 **8.1.2 Protocol**

1073 Figure 5 (non-normative) illustrates the overall logical flow of generic authority resolution.



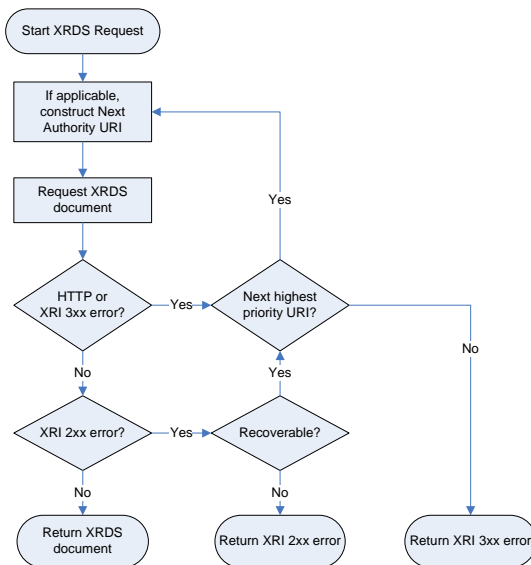
1074

1075 *Figure 5: Authority resolution flowchart.*

- 1076 Following are the normative requirements for behavior of an XRI resolver and an XRI resolution
 1077 service when performing generic XRI authority resolution:
- 1078 1. Each request for an XRDS document using HTTP(S) MUST conform to the requirements
 1079 in section 8.1.3.
 - 1080 2. For errors in XRDS document resolution requests, a resolver MUST implement failover
 1081 handling as specified in section 8.1.4.
 - 1082 3. The resolver MUST be preconfigured with or have a means of obtaining the XRDS
 1083 document describing the community root authority for the XRI to be resolved as defined
 1084 in section 8.1.5.
 - 1085 4. The resolver MAY obtain the XRDS document describing the community root authority by
 1086 requesting a self-describing XRDS document as defined in section 8.1.6.
 - 1087 5. Resolution of each subsegment in the Authority String after the community root
 1088 subsegment MUST proceed in subsegment order (left-to-right) using fully qualified
 1089 subsegment values as defined in section 8.1.7.
 - 1090 6. Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as
 1091 defined in section 8.1.8.
 - 1092 7. For each iteration of the authority resolution process, the next authority resolution service
 1093 endpoint MUST be selected as specified in section 8.1.9.
 - 1094 8. For each iteration of the authority resolution process, an HTTP(S) URI called the Next
 1095 Authority URI MUST be constructed as specified in section 8.1.10.
 - 1096 9. A resolver MAY request that a recursing authority resolution service perform resolution of
 1097 multiple subsegments as defined in section 8.1.11.
 - 1098 10. For each iteration of the authority resolution process, a resolver MUST perform Redirect
 1099 and Ref processing as specified in section 11. Note that if Redirect and Ref processing is
 1100 successful, it will result in a nested XRDS document as specified in section 11.4.

1101 **8.1.3 Requesting an XRDS Document using HTTP(S)**

1102 Figure 6 (non-normative) illustrates the logical flow for requesting an XRDS document.



1103
 1104 *Figure 6: XRDS request flowchart.*

1105 Following are the normative requirements for an XRI resolver and an XRI resolution service when
1106 requesting an XRDS document:

- 1107 1. Each resolution request MUST be an HTTP(S) GET to the Next Authority URI and MUST
1108 contain an Accept header with the media type identifier defined in Table 11. Note that in
1109 XRI authority resolution, this Accept header is NOT interpreted as an XRI resolution input
1110 parameter, but simply as the media type being requested from the server. This differs
1111 from XRI proxy resolution, where the Accept header MAY be used to specify the Service
1112 Media Type resolution parameter. See section 10.4.
- 1113 2. The ultimate HTTP(S) response from an authority resolution service to a successful
1114 resolution request MUST contain either: a) a 2XX response with a valid XRDS document
1115 containing an XRD element for each authority subsegment resolved, or b) a 304
1116 response signifying that the cached version on the resolver is still valid (depending on the
1117 client's HTTP(S) request). There is no restriction on intermediate redirects (i.e., 3XX
1118 result codes) or other result codes (e.g., a 100 HTTP response) that eventually result in a
1119 2XX or 304 response through normal operation of [RFC2616].
- 1120 3. The HTTP(S) response from an authority resolution service MUST return the media type
1121 requested by the resolver. The response SHOULD NOT include any media type
1122 parameters supplied by the resolver in the request. If the resolver receives such
1123 parameters in the response, the resolver MUST ignore them and do its own independent
1124 verification that the response fulfills the requested parameters.
- 1125 4. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in
1126 the resolution process. In this case, the resolver MUST implement failover handling as
1127 specified in section 8.1.4.
- 1128 5. If all authority resolution service endpoints fail, the resolver SHOULD return the
1129 appropriate error code and context message as specified in section 14. In recursing
1130 resolution, such an error MUST be returned by the recursing authority resolution service
1131 to the resolver as specified in section 14.4.
- 1132 6. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section
1133 15. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent
1134 possible to maintain the efficiency and scalability of the HTTP-based resolution system.
1135 The recommended use of HTTP caching headers is described in more detail in section
1136 15.2.1.

1137 8.1.4 Failover Handling

Comment [DSR17]: This section is new in ED06.

1138 XRI infrastructure has the same requirements as DNS infrastructure for stability, redundancy, and
1139 network performance. This means XRI authority and proxy resolution services are subject to the
1140 same requirements as DNS nameservers. For example:

- 1141 • Critical authority or proxy resolution servers SHOULD be operated from a minimum of two
1142 physically separate network locations to prevent a single point of failure.
- 1143 • Authority or proxy resolution servers handling heavy loads SHOULD operate from multiple
1144 servers and take advantage of load balancing technologies.

1145 However such capabilities are only effective if resolvers or other client applications implement
1146 proper failover handling. Because XRI resolution takes place at a layer above DNS resolution,
1147 resolvers have two ways to discover additional network endpoints at which authority or proxy
1148 resolution services are available.

- 1149 • *DNS round robin/failover*: The domain name of an authority resolution service endpoint URI
1150 may be associated with more than one IP address.
- 1151 • *XRI round robin/failover*: In a self-describing XRDS document (section 8.1.6), an XRI
1152 authority may publish multiple URI elements for its authority resolution service endpoint, or
1153 multiple authority resolution service endpoints, or both.

- 1154 To take advantage of both these options, the following rules apply to failover handling:
- 1155 1. A resolver SHOULD first try an alternate IP address for the current authority resolution
1156 service endpoint if DNS round robin is used.
 - 1157 2. If all alternate IP addresses fail, a resolver MUST try the next highest priority authority
1158 resolution URI in the current, if available.
 - 1159 3. If all URIs in the current authority resolution service endpoint fail, a resolver MUST try the
1160 next highest priority authority resolution service endpoint, if available, until all authority
1161 resolution service endpoints are exhausted.
 - 1162 4. A resolver SHOULD only return an error if all network endpoints associated with the
1163 authority resolution service fail to respond.

1164

1165 **IMPORTANT:** These same rules apply to any client or consuming application using an XRI proxy
1166 resolver, or else the proxy resolver can become another point of failure.

1167 One final consideration: DNS caching mechanisms should respect the TTL settings in DNS,
1168 however varying software languages and frameworks handle DNS caching differently. It is
1169 RECOMMENDED to check the default settings to ensure that a library or application is not
1170 caching DNS results indefinitely.

1171 8.1.5 Community Root Authorities

1172 Identifier management policies are defined on a community-by-community basis. For XRI
1173 authorities, the resolution community is specified by the first (leftmost) subsegment of the
1174 authority segment of the XRI. This is referred to as the *community root authority*. When a
1175 resolution community chooses to create a new community root authority, it SHOULD define
1176 policies for assigning and managing identifiers under this authority. Furthermore, it SHOULD
1177 define what resolution protocol(s) may be used for these identifiers.

1178 For an XRI authority, the community root may be either a global context symbol (GCS) character
1179 or top-level cross-reference as specified in section 2.2.1.1 of [XRISyntax]. In either case, the
1180 corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution
1181 service endpoints for that community.

1182 The community root authority SHOULD publish a self-describing XRDS document as defined in
1183 section 8.1.6. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as
1184 the community's root authority resolution service endpoints. This community root XRDS
1185 document, or its location, must be known *a priori* and is part of the configuration of an XRI
1186 resolver, similar to the specification of root DNS servers for a DNS resolver. Note that is not
1187 strictly necessary to publish this information in an XRDS document—it may be supplied in any
1188 format that enables configuration of the XRI resolvers in the community. However publishing a
1189 self-describing XRDS document at a known location simplifies this process and enables dynamic
1190 configuration of community resolvers.

1191 It is also a recommended best practice for a community root XRDS document to contain:

- 1192 • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- 1193 • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML
1194 trusted resolution is supported.
- 1195 • Both of the above if HTTPS+SAML trusted resolution is supported.
- 1196 • The service endpoints and supported media types of the community's XRI proxy resolver(s) if
1197 proxy resolution is supported.

1198 For a list of public community root authorities and the locations of their community root XRDS
1199 documents, see the Wikipedia entry on XRI [WikipediaXRI].

Comment [DSR18]: OPEN ISSUE:
This is a proposed solution to the
longstanding question of where the
spec might point to such references.

1200 8.1.6 Self-Describing XRDS Documents

1201 An identifier authority MAY publish a self-describing XRDS document, i.e., one produced by the
1202 same identifier authority that it describes. A resolver MAY request a self-describing XRDS
1203 document from a target identifier authority using either of two methods:

- 1204 1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution
1205 service endpoint, it may use the resolution protocol specified in section 5 to request an
1206 XRDS document directly from this HTTP(S) URI(s). This HTTP(S) URI may be known a
1207 priori (as is often the case with community root authorities, above), or it may be
1208 discovered from other identifier authorities via the resolution protocols defined in this
1209 specification.
- 1210 2. If the resolver knows: a) an XRI of the target authority as a community root authority, and
1211 b) the location of a proxy resolver configured for this community root authority, it may use
1212 the proxy resolution protocol specified in section 10 to query the proxy resolver for the
1213 community root authority XRI. This query MUST include only a single subsegment
1214 identifying the community root authority and MUST NOT include any additional
1215 subsegments.

1216 An example of the first method, if a identifier authority had an authority resolution service
1217 endpoint at `http://example.com/auth-res-service/`, would be to issue an HTTP(S) GET
1218 request to that URI with an Accept header specifying the content type
1219 `application/xrds+xml`. See section 6.3 for more details.

1220 An example of the second method, if a identifier authority had the community root authority
1221 identifier `xri://(example)` and was registered with the XRI proxy resolver
1222 `http://xri.example.com/`, would be to issue an HTTP(S) GET request to the following URI:

1223 `http://xri.example.com/(example)?_xrd_r=application/xrds+xml`

1224 See section 10 for more details.

1225 Note that a proxy resolver may use the first method to publish its own self-describing XRDS
1226 document at the HTTP(S) URI(s) for its proxy resolution service.

1227 **IMPORTANT:** A self-describing XRDS document MUST only be issued by an identifier authority
1228 when describing itself. It MUST NOT be included when the identifier authority is describing a
1229 different identifier authority. In the latter case the self-describing XRDS document for the
1230 community root authority is implicit. It MAY be explicitly requested by the resolver if needed for
1231 dynamic configuration of the resolver or trust verification of other XRI resolution chains.

1232 8.1.7 Qualified Subsegments

1233 A qualified subsegment is defined by the productions whose names start with "xri-subseg" in
1234 section 2.2.3 of **[XRISyntax]** including the leading syntactic delimiter ("*" or "!"). A qualified
1235 subsegment MUST include the leading syntatic delimiter even if it was optionally omitted in the
1236 original XRI (see section 2.2.3 of **[XRISyntax]**).

1237 If the first subsegment of an XRI authority is a GCS character and the following subsegment does
1238 not begin with a "*" (indicating a reassignable subsegment) or a "!" (indicating a persistent
1239 subsegment), then a "*" is implied and MUST be added when constructing the qualified
1240 subsegment as specified in section 0. Table 12 and Table 13 illustrate the differences between
1241 parsing a reassignable subsegment following a GCS character and parsing a cross-reference,
1242 respectively.

1243

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

1244 *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

1245 *Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.*

1246 **8.1.8 Cross-References**

1247 Any subsegment within an XRI authority segment may be a cross-reference (see section 2.2.2 of
 1248 **[XRI Syntax]**). Cross-references are resolved identically to any other subsegment because the
 1249 cross-reference is considered opaque, i.e., the value of the cross-reference (including the
 1250 parentheses) is the literal value of the subsegment for the purpose of resolution.

1251 Table 14 provides several examples of resolving cross-references. In these examples,
 1252 subsegment "lb" resolves to a Next Authority Service Endpoint URI of "http://example.com/xri-
 1253 authority/" and recursing authority resolution is not being requested.
 1254

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)

1255 *Table 14: Examples of the Next Authority URIs constructed using different types of cross-references.*

1256 **8.1.9 Selection of the Next Authority Resolution Service Endpoint**

1257 For each iteration of authority resolution, the resolver MUST select the next authority resolution
 1258 service endpoint from the current XRD as specified in section 12. For generic authority resolution,
 1259 this selection process MUST use the parameters specified in Table 11. For trusted authority
 1260 resolution, this selection process MUST use the parameters specified in Table 15, Table 16, or
 1261 Table 17. In all cases, an explicit match on the `xrd:XRD/xrd:Service/xrd:Type` element is
 1262 REQUIRED, so during authority resolution, a resolver MUST set the `nodefault` parameter to a

1263 value of `nodefault=type` in order to override selection of a default service endpoint as
1264 specified in section 12.3.2.

1265 8.1.10 Construction of the Next Authority URI

1266 Once the next authority resolution service endpoint is selected, the resolver MUST construct a
1267 URI for the next HTTP(S) request, called the Next Authority URI, by concatenating two strings as
1268 specified in this section.

1269 The first string is selected from the next authority resolution service endpoint selected as
1270 specified in section 8.1.9. The resolver MUST select the highest priority URI of the highest priority
1271 authority resolution service endpoint. Next, the resolver MUST apply the service endpoint URI
1272 construction algorithm based the value of the `append` attribute as defined in section 12.7.

1273 This fully constructed URI is called the *Next Authority Service Endpoint URI*. If this URI does not
1274 end with a forward slash ("`/`"), one MUST be appended before proceeding.

1275 The second string is called the *Next Authority String* and it consists of either:

- 1276 • The next fully qualified subsegment to be resolved (see section 8.1.7), or
- 1277 • In the case of recursing resolution, the next fully qualified subsegment to be resolved plus
1278 any additional subsegments for which recursing resolution is requested (see section 8.1.8).

1279 The final step is to append the Next Authority String to the path component of the Next Authority
1280 Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

1281 Construction of the Next Authority URI is more formally described in this pseudocode for
1282 resolving a "next-auth-string" via a "next-auth-sep-uri":

```
1283 if (path portion of next-auth-sep-uri does not end in "/"):  
1284     append "/" to path portion of next-auth-sep-uri  
1285  
1286 if (next-auth-string is not preceded with "*" or "!" delimiter):  
1287     prepend "*" to next-auth-string  
1288  
1289 append uri-escape(next-auth-string) to path of next-auth-sep-uri
```

1290 8.1.11 Recursing Authority Resolution

1291 If an authority resolution service offers recursing resolution, an XRI resolver MAY request
1292 resolution of multiple authority subsegments in one transaction. If a resolver makes such a
1293 request, the responding authority resolution service MAY perform the additional recursing
1294 resolution steps requested. In this case the recursing authority resolution service acts as a
1295 resolver to the other authority resolution service endpoints that need to be queried. Alternatively,
1296 the recursing authority resolution service may retrieve XRDs from its local cache until it reaches a
1297 subsegment whose XRD is not locally cached, or it may simply recurse only as far as it is
1298 authoritative. If an authority resolution service performs any recursing resolution, it MUST return
1299 an ordered list of `xrd:XRDS` elements (and nested `xrd:XRDS` elements if Redirects or Refs are
1300 followed as specified in section 11) in an `xrd:XRDS` document for all subsegments resolved as
1301 defined in section 7.2.1.

1302 A recursing authority resolution service MAY resolve fewer subsegments than requested by the
1303 resolver. The recursing authority resolution service is under no obligation to resolve more than
1304 the first subsegment (for which it is, by definition, authoritative).

1305 If the recursing authority resolution service does not resolve the entire set of subsegments
1306 requested, the resolver MUST continuing the authority resolution process itself. At any stage,
1307 however, the resolver MAY request the next authority resolution service to recursively resolve any
1308 remaining subsegments.

1309 8.2 IRI Authority Resolution

1310 From the standpoint of generic authority resolution, an IRI authority segment represents either a
1311 DNS name or an IP address at which an XRDS document describing the authority may be
1312 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET
1313 request to a URI constructed from the IRI authority segment. The resulting XRDS document can
1314 then be consumed in the same manner as one obtained using XRI authority resolution.

1315 While the use of IRI authorities provides backwards compatibility with the large installed base of
1316 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of
1317 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities
1318 are not recommended for new deployments of XRI identifiers.

1319 This section defines IRI authority resolution as a simple extension to the XRI authority resolution
1320 protocol defined in the preceding section.

1321 8.2.1 Service Type and Media Type

1322 Because IRI authority resolution takes place at a level “below” XRI authority resolution, it cannot
1323 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority
1324 resolution uses the same media type as generic XRI authority resolution.

1325 8.2.2 Protocol

1326 Following are the normative requirements for IRI authority resolution that differ from generic XRI
1327 authority resolution:

1328 1. The next authority URI is constructed by extracting the entire IRI authority segment and
1329 prepending the string “http://”. See the exception in section 8.2.3.

1330 2. The HTTP GET request MUST include an HTTP Accept header containing only the
1331 following:

1332 `Accept: application/xrds+xml`

1333 3. The HTTP GET request MUST have a Host: header (as defined in section 14.23 of
1334 **[RFC2616]**) containing the value of the IRI authority segment.

1335 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document
1336 containing the XRD describing that authority.

1337 5. The responding server MUST use the value of the Host header to populate the
1338 `xrd:XRD/xrd:Query` element in the resulting XRD. For example:

1339 `Host: example.com`

1340 Note that because IRI authority resolution is required to process the entire IRI authority segment
1341 in a single step, recursing authority resolution does not apply.

1342 8.2.3 Optional Use of HTTPS

1343 Section 9 of this specification defines trusted resolution only for XRI authorities. Trusted
1344 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to
1345 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY
1346 use HTTPS as the access protocol for retrieving the authority’s XRD. If the resolver is satisfied,
1347 via transport level security mechanisms, that the response is from the expected IRI authority, the
1348 resolver may consider this an HTTPS trusted resolution response as defined in section 9.1.

1349 9 Trusted Authority Resolution

1350 This section defines three options for performing trusted XRI authority resolution as an extension
1351 of the generic XRI authority resolution service defined in section 8.1—one using HTTPS, one
1352 using SAML assertions, and one using both.

1353 9.1 HTTPS

1354 This option for trusted authority resolution is a very simple addition to generic authority resolution
1355 in which all communication with authority resolution service endpoints is carried out over HTTPS.
1356 This provides transport-level security and server authentication, however it does not provide
1357 message-level security or a means for a responder to provide different responses for different
1358 requestors.

1359 9.1.1 Service Type and Service Media Type

1360 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

1361 *Table 15: Service Type and Service Media Type values for HTTPS trusted authority resolution.*

1362 An HTTPS trusted resolution service endpoint advertised in an XRDS document MUST use the
1363 Service Type identifier and Service Media Type identifier (including the `https=true` parameter)
1364 defined in Table 15. In addition, the identifier authority MUST use an HTTPS URI as the value of
1365 the `xrd:URI` element(s) for this service endpoint.

1366 9.1.2 Protocol

1367 Following are the normative requirements for HTTPS trusted authority resolution that differ from
1368 generic XRI authority resolution (section 8.1):

- 1369 1. All authority resolution service endpoints MUST be selected using the values defined in
1370 Table 15.
- 1371 2. All authority resolution requests including the starting request to a community root
1372 authority MUST use the HTTPS protocol as defined in [RFC2818]. This includes all
1373 authority resolution requests resulting from Redirect and Ref processing as defined in
1374 section 11. A successful HTTPS response MUST be received from each authority in the
1375 resolution chain or the resolver MUST output an error.
- 1376 3. All authority resolution requests MUST contain an HTTPS Accept header with the media
1377 type identifier defined in Table 15 (including the `https="true"` parameter).
- 1378 4. If the resolver finds that an authority in the resolution chain does not support HTTPS, the
1379 resolver MUST return a 23x error as defined in section 14.

1380 9.2 SAML

1381 In SAML trusted resolution, the resolver requests a content type of `application/xrds+xml;`
1382 `saml=true` and the authority resolution service responds with an XRDS document containing an
1383 XRD with an additional element—a digitally signed SAML [SAML] assertion that asserts the
1384 validity of the containing XRD. SAML trusted resolution provides message integrity but does not
1385 provide confidentiality. The latter may be achieved by combining it with HTTPS as defined in

1386 section 9.3. Message confidentiality may also be achieved with other security protocols used in
1387 conjunction with this specification. SAML trusted resolution also does not provide a means for an
1388 authority to provide different responses for different requestors; client authentication is explicitly
1389 out-of-scope for version 2.0 of XRI resolution.

1390 9.2.1 Service Type and Service Media Type

1391 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

1392 *Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.*

1393 A SAML trusted resolution service endpoint advertised in an XRD document MUST use the
1394 Service Type identifier and Service Media Type identifier defined in Table 16 (including the
1395 `saml=true` parameter). In addition, for transport security the identifier authority SHOULD offer at
1396 least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1397 9.2.2 Protocol

1398 9.2.2.1 Client Requirements

1399 For a resolver, trusted resolution is identical to the generic resolution protocol (section 8.1) with
1400 the addition of the following requirements:

- 1401 1. All authority resolution service endpoints MUST be selected using the values defined in
1402 Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an
1403 authority unless the authority advertises a resolution service endpoint matching these
1404 values.
- 1405 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is
1406 RECOMMENDED for confidentiality.
- 1407 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the
1408 media type identifier defined in Table 16 (including the `saml=true` parameter). This is
1409 the media type of the request Resolution Output Format. (Clients willing to accept either
1410 generic or trusted responses may use a combination of media type identifiers in the
1411 Accept header as described in section 14.1 of [RFC2616]. Media type identifiers
1412 SHOULD be ordered according to the client's preference for the media type of the
1413 response. If a client performing generic authority resolution receives an XRD containing
1414 SAML elements, it is NOT REQUIRED to validate the signature or perform any
1415 processing of these elements.)
- 1416 4. A resolver MAY request recursing authority resolution of multiple subsegments as
1417 defined in section 9.2.3.
- 1418 5. The resolver MUST individually validate each XRD in the resolution chain according to
1419 the rules defined in section 9.2.4. When `xrd:XRD` elements come both from freshly-
1420 retrieved XRD documents and from a local cache, a resolver MUST ensure that these
1421 requirements are satisfied each time a resolution request is performed.

1422 9.2.2.2 Server Requirements

1423 For an authority resolution service, trusted resolution is identical to the generic resolution protocol
1424 (section 8.1) with the addition of the following requirements:

- 1425 1. The HTTP(S) response to a trusted resolution request MUST include a content type of
1426 "application/xrds+xml;trust=saml".

- 1427 2. The XRDS document returned by the resolution service MUST contain a
 1428 `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid
 1429 per the processing rules described by [SAML].
- 1430 3. The `saml:Assertion` element MUST contain a valid enveloped digital signature as
 1431 defined by [XMLDSig] and as constrained by section 5.4 of [SAML].
- 1432 4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML
 1433 assertion. Specifically, the signature MUST contain a single
 1434 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference
 1435 MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML
 1436 assertion. The `URI` reference MUST NOT be empty and it MUST refer to the identifier
 1437 contained in the `xrd:XRD/@xml:id` attribute.
- 1438 5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain
 1439 a `ds:KeyInfo` element. If this element is included, it MUST describe the key used to
 1440 verify the digital signature element. However, because the signing key is known in
 1441 advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the
 1442 `ds:Signature` element of the SAML assertion.
- 1443 6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST
 1444 match the XRI authority subsegment requested by the client.
- 1445 7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match
 1446 the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD
 1447 advertising availability of trusted resolution service from this authority as required in
 1448 section 9.2.5.
- 1449 8. The `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
 1450 present and equal to the `xrd:XRD/xrd:Query` element.
- 1451 9. The `NameQualifier` attribute of the
 1452 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
 1453 present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.
- 1454 10. There MUST be exactly one `saml:AttributeStatement` present in the
 1455 `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute`
 1456 element with a `Name` attribute of "`xri://$xrd*($v*2.0)`". This `saml:Attribute`
 1457 element MUST contain exactly one `saml:AttributeValue` element whose text value
 1458 is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that is the immediate
 1459 parent of the `saml:Assertion` element.

Comment [DSR19]: TODO: confirm the Name attribute value (Peter)

1460 9.2.3 Recursing Authority Resolution

1461 If a resolver requests trusted resolution of multiple authority subsegments (see section 8.1.8), a
 1462 recursing authority resolution service SHOULD attempt to perform trusted resolution on behalf of
 1463 the resolver as described in this section. However if the resolution service is not able to obtain
 1464 trusted XRDS for one or more additional recursing subsegments, it SHOULD return only the
 1465 trusted XRDS it has obtained and allow the resolver to continue.

1466 9.2.4 Client Validation of XRDS

1467 For each XRD returned as part of a trusted resolution request, the resolver MUST validate the
 1468 XRD according to the rules defined in this section.

- 1469 1. The `xrd:XRD/saml:Assertion` element MUST be present.
- 1470 2. This assertion MUST valid per the processing rules described by [SAML].
- 1471 3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by
 1472 [XMLDSig] and constrained by Section 5.4 of [SAML].

- 1473 4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML
1474 assertion. Specifically, the signature MUST contain a single
1475 `ds:SignedInfo/ds:Reference` element, and the URI attribute of this reference
1476 MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent
1477 of the signed SAML assertion.
- 1478 5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo`
1479 element, the resolver MAY reject the signature if this key does not match the signer's
1480 expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor
1481 that was used to describe the current authority. See section 9.2.5.
- 1482 6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose
1483 resolution resulted in the current XRD.
- 1484 7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1485 `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability
1486 of trusted resolution service from this authority as required in section 9.2.5.
- 1487 8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1488 `NameQualifier` attribute of the
1489 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1490 9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the
1491 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1492 10. There MUST exist exactly one
1493 `xrd:XRD/saml:Assertion/saml:AttributeStatment` with exactly one
1494 `saml:Attribute` element that has a `Name` attribute of "`xri://$xrd*($v*2.0)`". This
1495 `saml:Attribute` element must have exactly one `saml:AttributeValue` element
1496 whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that
1497 is the immediate parent of the signed SAML assertion.

1498 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result
1499 MUST NOT be considered a valid trusted resolution response as defined by this specification.
1500 Note that this does not preclude a resolver from considering alternative resolution paths. For
1501 example, if an XRD advertising SAML trusted resolution service has two or more
1502 `xrd:XRD/xrd:Service/xrd:URI` elements and the response from one service endpoint fails
1503 to meet the requirements above, the client MAY repeat the validation process using the second
1504 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as
1505 defined by this document and SAML trusted resolution may continue.

1506 [If all SAML trusted resolution paths fail, the resolver MUST return the appropriate 23x trusted](#)
1507 [resolution error as defined in section 14.](#)

1508 9.2.5 Correlation of ProviderID and KeyInfo Elements

1509 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at
1510 least one unique persistent service provider identifier expressed in the
1511 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority
1512 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI
1513 authority. A ProviderID may be any valid URI that meets these requirements of persistence and
1514 uniqueness. Examples of appropriate URIs include fully persistent XRIs expressed in URI-normal
1515 form as defined by **[XRISyntax]** and URNs as defined by **[RFC2141]**.

1516 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in
1517 an XRD advertising trusted authority resolution service with the response received from a trusted
1518 resolution service endpoint. If the signed XRD response contains the same ProviderID as the
1519 XRD used to advertise a service, and the resolver has reason to trust the signature, the resolver
1520 can trust that the XRD response has not been maliciously replaced with another XRD.

1521 There is no defined discovery process for the ProviderID for a community root authority; it must
 1522 be published in a self-describing XRDS document (or other equivalent description—see sections
 1523 8.1.5 and 8.1.6) and verified independently. Once the community root XRDS document is known,
 1524 the ProviderID for delegated XRI authorities within this community MAY be discovered using the
 1525 `xrd:XRDS/Service/xrd:ProviderID` element of authority resolution service endpoints.
 1526 This trust mechanism may also be used for other services offered by an authority.

1527 In addition, the metadata necessary for SAML trusted authority resolution or other SAML [SAML]
 1528 interactions MAY be discovered using the `ds:KeyInfo` element (section 4.2.) Again, if this
 1529 element is present in an XRD advertising SAML authority resolution service (or any other
 1530 service), and the client has reason to trust this XRD, the client MAY use the associated
 1531 ProviderID to correlate the contents of this element with a signed response.

1532 To assist resolvers in using this key discovery mechanism, it is important that trusted authority
 1533 resolution services be configured to sign responses in such a way that the signature can be
 1534 verified using the correlated `ds:KeyInfo` element. For more information, see [SAML].

1535 9.3 HTTPS+SAML

1536 9.3.1 Service Type and Service Media Type

1537 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>

1538 *Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.*

1539 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST
 1540 use the Service Type identifier and Service Media Type identifier defined in Table 17 (including
 1541 the `https=true` and `saml=true` parameters). In addition, the identifier authority MUST use an
 1542 HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1543 9.3.2 Protocol

1544 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- 1545 1. All authority resolution service endpoints MUST be selected using the values defined in
 1546 Table 17.
- 1547 2. All authority resolution requests and responses, including the starting request to a
 1548 community root authority, MUST conform to both the requirements of the HTTPS trusted
 1549 resolution protocol defined in section 9.1 and the SAML trusted resolution protocol
 1550 defined in section 9.2.
- 1551 3. All authority resolution requests MUST contain an HTTPS Accept header with the media
 1552 type identifier defined in Table 17 (including both the `https=true` and `saml=true`
 1553 parameters). This MUST be interpreted as the value of the Resolution Output Format
 1554 input parameter.
- 1555 4. If the resolver finds that an authority in the resolution chain does not support both HTTPS
 1556 and SAML, the resolver MUST return a 23x error as defined in section 14.

1557

10 Proxy Resolution

1558
1559
1560
1561
1562

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

1563

Proxy resolution is useful for many reasons:

1564
1565

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

1566
1567
1568

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

1569
1570
1571

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

1572

10.1 Service Type and Media Types

1573

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

1574

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

1575
1576
1577
1578
1579

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

1580
1581
1582
1583
1584
1585
1586

It may appear to be of limited value to advertise proxy resolution service in an XRDS document if a resolver must already know how to perform local XRI resolution in order to retrieve this document. However advertising a proxy resolution service in the XRDS document for a community root authority (sections 8.1.3 and 8.1.6) can be very useful for applications that need to consume XRI proxy resolution services or automatically generate HXRIs for resolution by non-XRI-aware clients in that community. Those applications may discover the current URI(s) and media type capabilities of a proxy resolver from this source.

1587

10.2 HXRIs

Comment [DSR20]: In the future this section will move to XRI Syntax 3.0.

1588
1589
1590

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

1591
1592

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

1593
1594

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

1595 To make this syntax as simple as possible for XRI-aware processors or search agents to
1596 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority segment that
1597 begins with the domain name segment "xri.". The QXRI is then appended as the entire local
1598 path (and query component, if present). The QXRI MUST NOT include the "xri://" prefix and
1599 MUST be in URI-normal form as defined in [XRISyntax]. (If a proxy resolver receives an HXRI
1600 containing a QXRI beginning with an "xri://" prefix, it SHOULD follow Postel's Law¹ by
1601 removing it before continuing.) In essence, the proxy resolver URI (including the forward slash
1602 after the domain name) serves as a machine-readable prefix for an absolute XRI in URI-normal
1603 form.

1604 The normative ABNF for an HXRI is defined below based on the *ireg-name*, *xri-hier-part*,
1605 and *iquery* productions defined in [XRISyntax]. Authors whose XRIs need to be understood by
1606 non-XRI-aware clients SHOULD publish them as HTTP URIs conforming to this HXRI production.

```
1607 HXRI           = proxy-resolver "/" QXRI
1608 proxy-resolver = ( "http://" / "https://" ) proxy-reg-name
1609 proxy-reg-name = "xri." ireg-name
1610 QXRI          = xri-hier-part [ "?" i-query ]
```

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

1611 URI processors that recognize XRIs SHOULD interpret the local part of an HTTP or HTTPS URI
 1612 (the path segments and optional query segment) as an XRI provide that: a) it conforms to this
 1613 ABNF, and b) the first segment of the path conforms to the xri-authority or 1authority productions
 1614 in [XRISyntax].

1615 For references to communities that offer public XRI proxy resolution services, see the Wikipedia
 1616 entry on XRI [WikipediaXRI].

Comment [DSR21]: As above, this is a proposed to the issue of how the spec can provide a neutral reference to public resolution communities.

1617 10.3 HXRI Query Parameters

1618 There are a total of five logical input parameters to XRI authority resolution and service endpoint
 1619 selection as defined in section 7.1:

- 1620 1. QXRI – the entire original XRI for which resolution is requested.
- 1621 2. Path String – the path component of the QXRI.
- 1622 3. Resolution Output Format – the requested media type of the resolution response (see
 1623 section 7.1.2).
- 1624 4. Service Type – the type of service requested for service endpoint selection (see section
 1625 7.1.3).
- 1626 5. Service Media Type – the type of media requested for service endpoint selection (see
 1627 section 7.1.4).

1628 In proxy resolution, these parameters are bound to an HTTP(S) interface using the conventional
 1629 web model of encoding them in an HTTP(S) URI, which in this case is an HXRI. The binding of
 1630 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

1631 *Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.*

1632 Following are the rules for the use of the parameters specified in Table 19.

- 1633 1. The QXRI MUST be normalized as specified in section 10.2.
- 1634 2. If the original QXRI has an existing query component, the HXRI query parameters MUST
 1635 be appended to that query component. (Note that the query parameter names in Table
 1636 19 were chosen to minimize the probability of collision with any other existing query
 1637 parameter names.) After proxy resolution, the HXRI query parameters MUST
 1638 subsequently be removed from the QXRI query component. The existing QXRI query
 1639 component MUST NOT be altered in any other way, i.e., it must be passed through with
 1640 no changes in parameter order, escape encoding, etc.
- 1641 3. If the original QXRI does not have a query component, one MUST be added to pass any
 1642 HXRI query parameters. After proxy resolution, this query component MUST be entirely
 1643 removed.

- 1644 4. If the original QXRI had a null query component (only a leading question mark), or a
 1645 query component consisting of only question marks, *one additional leading question mark*
 1646 MUST be added before adding any HXRI query parameters. After proxy resolution, any
 1647 HXRI query parameters and exactly one leading question mark MUST be removed. See
 1648 the URI construction steps defined in section 12.6.
- 1649 5. Each HXRI query parameter MUST be delimited from other parameters by an ampersand
 1650 (“&”). Any occurrences of the character “&” within an input parameter (specifically the
 1651 Service Type value) MUST be percent encoded prior to input and decoded upon output.
- 1652 6. Each HXRI query parameter MUST be delimited from its value by an equals sign (“=”).
- 1653 7. In an HXRI query parameter, the character + and the percent-encoded sequence %2B
 1654 MUST be interpreted as the same character, therefore spaces in an HXRI query
 1655 parameter MUST NOT be encoded as + signs.
- 1656 8. If an HXRI query parameter includes one of the media type parameters defined in Table ← --- **Formatted: Bullets and Numbering**
 1657 6, it MUST be delimited from the HXRI query parameter with a semicolon, and this
 1658 semicolon MUST be percent-encoded using the sequence %3B. This prevents
 1659 misinterpretation of the semicolon character by a proxy resolver.
- 1660 9. Any XRIs or IRIs used as values of HXRI query parameters MUST be in URI-normal form
 1661 as defined in **[XRISyntax]**.
- 1662 10. If any HXRI query parameter name is included but its value is empty, the value of the
 1663 parameter MUST be considered null.

1664 10.4 HTTP(S) Accept Headers

1665 In proxy resolution, one XRI resolution input parameter, the Service Media Type (section 7.1.4)
 1666 MAY be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. The
 1667 following rules apply to this input:

- 1668 1. As described in section 14.1 of **[RFC2616]**, the Accept header content type MAY consist
 1669 of multiple media type identifiers. If so, the proxy resolver MUST choose only one to
 1670 accept. A proxy resolver client SHOULD order media type identifiers according to the
 1671 client's preference and a proxy resolver server SHOULD choose the client's highest
 1672 preference.
- 1673 2. If the value of the Accept header content type is null, this MUST be interpreted as the
 1674 value of the Service Media Type parameter.
- 1675 3. If the value of the Service Media Type parameter is explicitly set via the `_xrd_m` query
 1676 parameter in the HXRI (including to a null value), this MUST take precedence over any
 1677 value set via an HTTP(S) Accept header.

1678 10.5 Null Resolution Output Format

1679 Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a
 1680 resolution request where the Resolution Output Format input parameter value is null—either
 1681 because this parameter is absent or because it was explicitly set to null using the `_xrd_r` query
 1682 parameter.

1683 If the value of the Resolution Output Format value is null, a resolver MUST act as if the following
 1684 media type parameters had the following values: `https=false`, `saml=false`, `refs=true`,
 1685 `sep=true`, and `nodefault=false`. In addition, the output MUST be an HTTP(S) redirect as
 1686 defined in the following section.

1687 10.6 Outputs and HTTP(S) Redirects

1688 For all values of the Resolution Output Format parameter except null, a proxy resolver MUST
 1689 follow the output rules defined in section 7.2.

1690 If the value of the Resolution Output Format is null, and the output is not an error, a proxy
1691 resolver MUST follow the rules for output of a URI List as defined in section 7.2.3. However
1692 instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as
1693 an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service
1694 Media Type parameter.

1695 If the output is an error, a proxy resolver SHOULD return a human-readable error message as
1696 specified in section 14.4.

1697 This rule enables XRI proxy resolvers to serve clients that do not understand XRI syntax or
1698 resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the
1699 service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept
1700 header (if any) as described in section 10.4.

1701 **10.7 Differences Between Proxy Resolution Servers**

1702 An XRI proxy resolution request MAY be sent to any proxy resolver that will accept it. All XRI
1703 proxy resolvers SHOULD deliver uniform responses given the same QXRI and other input
1704 parameters. However because proxy resolvers may potentially need to make decisions about
1705 network errors, reference processing, and trust policies on behalf of the client they are proxying,
1706 and these decisions may be based on local policy, in some cases different proxy resolvers may
1707 return different results.

1708 **10.8 Combining Authority and Proxy Resolution Servers**

1709 The majority of DNS nameservers are recursing nameservers that answer both queries for which
1710 they are authoritative and queries which they must forward to other nameservers. The same rule
1711 applies to XRI architecture: in many cases the optimum configuration will be to combine an
1712 authority resolution service and a proxy resolution service in the same server. This server can
1713 publish a self-describing XRDS document (section 8.1.6) that advertises both its authority
1714 resolution and proxy resolution service endpoints. It can also optimize caching of XRDS for clients
1715 in its resolution community (see section 15.4).

1716 11 Redirect and Ref Processing

1717 As discussed in section 5.3, the purpose of the `xrd:Redirect` and `xrd:Ref` elements is to
 1718 enable identifier authorities to distribute and delegate management of XRDS documents. There
 1719 are two primary use cases for using multiple XRDS documents to describe the same resource:

- 1720 • One identifier authority needs to manage descriptions of the resource from different physical
 1721 locations on the network, e.g., in a registry, in a directory, on a webserver, etc. This is the
 1722 purpose of the `xrd:Redirect` element.
- 1723 • One identifier authority needs to delegate all or part of managing descriptions of the resource
 1724 to different identifier authorities, e.g., an individual may delegate responsibility for different
 1725 aspects of an XRDS to his/her spouse, school, employer, doctor, etc. This is the purpose of
 1726 the `xrd:Ref` element.

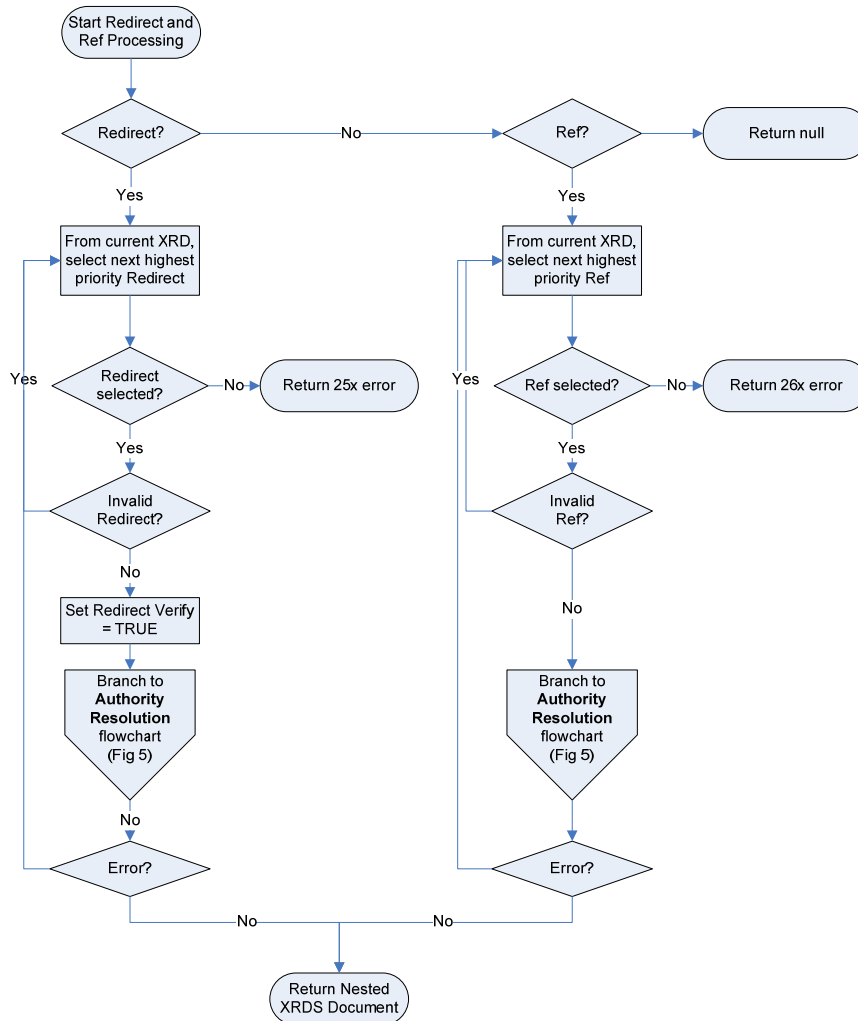
1727 Table 20 summarizes the similarities and differences between the `xrd:Redirect` and `xrd:Ref`
 1728 elements.

Requirement	Redirect	Ref
Must contain	HTTP(S) URI	XRI
Accepts the same <code>append</code> attribute as the <code>xrd:URI</code> element	Yes	No
Delegates to a different authority	No – just redirects to a different network location	Yes
Must include a subset of the synonyms available in the source XRD	Yes	No
Available at both authority level and service level	Yes	Yes
Processed automatically if present at the authority level	Yes	Yes
Successful resolution results in nested XRDS document	Yes	Yes
Required attribute of XRDS element for nested XRDS document	<code>redirect</code>	<code>ref</code>
Number of XRDS in nested XRDS document	1	1 or more

1729 *Table 20: Comparison of Redirect and Ref elements.*

1730 The combination of these two elements should enable identifier authorities to implement any set
 1731 of XRDS management policies.

1732 Figure 7 (non-normative) illustrates the logical flow of Redirect and Ref processing.



1733

1734 *Figure 7: Redirect and Ref processing flowchart.*

1735 This section contains the normative requirements for processing of `xrd:Redirect` and
 1736 `xrd:Ref` elements.

1737 11.1 Cardinality

1738 Redirect and Ref elements may be used both at the authority level (as a child of the `xrd:XRD`
 1739 element) and the service level (as a child of the `xrd:XRD/xrd:Service` element) within an
 1740 XRD. In both cases to simplify processing the XRD Schema (Appendix D) enforces the following
 1741 rules:

- 1742 • At the authority level, an XRD MAY contain one of two choices: zero-or-more
 1743 `xrd:Redirect` or zero-or-more `xrd:Ref` elements.

- 1744 • At the service level, an XRD MAY contain one of three choices: zero-or-more `xrd:URI`
1745 elements, zero-or-more `xrd:Redirect` elements, or zero-or-more `xrd:Ref` elements.

1746 11.2 Redirect Processing

1747 The purpose of the `xrd:Redirect` element is to enable an authority to redirect from one XRDS
1748 document managed in one network location (e.g., a registry) to another managed in a different
1749 network location by the same authority in (e.g., a web server, blog, etc.) It is similar to an
1750 HTTP(S) redirect, however it is managed at the XRDS document level rather than HTTP(S)
1751 transport level. Note that unlike a `Ref`, a `Redirect` does NOT delegate to a different XRI authority,
1752 but only to the same authority at a different network location.

1753 Following are the normative rules for processing of the `xrd:Redirect` element.

- 1754 1. If one or more `xrd:XRDXrd:Redirect` elements are present at the authority level of
1755 an XRD, they MUST be processed automatically before a resolver continues with
1756 authority resolution, performs service endpoint selection, or returns its final output. This
1757 rule applies iteratively to all XRDS documents resolved as a result of `Redirect`
1758 processing.
- 1759 2. If an `xrd:XRDXrd:Redirect` element is not present at the authority level of an XRD,
1760 but one or more `xrd:XRDXrd:Service/xrd:Redirect` elements are present in the
1761 highest priority selected service endpoint, they MUST be processed before a resolver
1762 completes service endpoint selection. If all `xrd:XRDXrd:Service/xrd:Redirect`
1763 elements fail in the highest priority selected service endpoint, the resolver MUST try the
1764 highest priority `xrd:XRDXrd:Service/xrd:Redirect` element in the next highest
1765 priority selected service endpoint, and continue until all priority
1766 `xrd:XRDXrd:Service/xrd:Redirect` elements in all selected service endpoints
1767 are exhausted. This rule applies iteratively to all XRDS documents resolved as a result of
1768 `Redirect` processing.
- 1769 3. To process a `Redirect` at either the authority or service level, the resolver MUST begin by
1770 selecting the highest priority `xrd:XRDXrd:Redirect` element.
- 1771 4. If the value of the resolution media type parameter `https` is `false`, or the parameter is
1772 absent or empty, the value of the selected `xrd:Redirect` element MUST be a valid
1773 HTTPS URI. If not, the resolver MUST select the next highest priority `xrd:Redirect`
1774 element. If all instances of this element fail, the resolver MUST return the error 251
1775 `INVALID_REDIRECT` in the XRD containing the `Redirect` as defined in section 14.
- 1776 5. If the value of the resolution media type parameter `https` is `true`, the value of the
1777 selected `xrd:Redirect` element MUST be a valid HTTPS URI. If not, the resolver
1778 MUST select the next highest priority `xrd:Redirect` element. If all instances of this
1779 element fail, the resolver MUST return the error 252 `INVALID_HTTPS_REDIRECT` in the
1780 XRD containing the `Redirect` as defined in section 14.
- 1781 6. Once a valid an `xrd:Redirect` element has been selected, if the
1782 `xrd:XRDXrd:Redirect` element includes the `append` attribute, the resolver MUST
1783 construct the final HTTP(S) URI as defined in section 12.7.
- 1784 7. The resolver MUST request a new XRDS document from the final HTTP(S) URI using the
1785 protocol defined in section 6.3. If the Resolution Output Format is an XRDS document,
1786 the resolver MUST embed a nested XRDS document containing an XRD representing
1787 the `Redirect` as specified in section 11.4.
- 1788 8. If resolution of an `xrd:Redirect` element fails, the resolver MUST report the error in
1789 the XRD contained in the nested XRDS document using the status codes defined in
1790 section 14. (One nested XRDS document will be added for each `Redirect` attempted by
1791 the resolver.) The resolver MUST then select the next highest priority `xrd:Redirect`

- 1792 element and repeat step 7. For more details, see section 11.5, *Recursion and*
1793 *Backtracking*.
- 1794 9. If resolution of all `xrd:Redirect` elements fails, the resolver MUST stop and return the
1795 error in the requested Resolution Output Format as defined in section 7.2.
- 1796 10. If resolution succeeds, the resolver MUST verify that all XRD synonym elements
1797 contained in the new XRD (as specified in section 5.2) are equivalent to or a subset of
1798 those contained in the XRD containing the Redirect. If not, the resolver MUST stop and
1799 immediately return the error 253 `REDIRECT_VERIFY_FAILED` in the XRD where the
1800 error occurred as defined in section 14.
- 1801 11. If the value of the resolution media type parameter `saml` is `true`, the resolver MUST
1802 verify the signature on the XRD as specified in section 9.2.4. If signature verification fails,
1803 the resolver MUST stop and report an error as defined in section 9.2.4.
- 1804 12. If synonym verification succeeds, further authority resolution or service endpoint selection
1805 MUST continue based on the new XRD.

1806 11.3 Ref Processing

1807 The purpose of the `xrd:Redirect` element is to enable one authority to delegate management
1808 of all or part of an XRDS document to another authority. For example, an individual might
1809 delegate management of all or portions of an XRDS document to his/her spouse, school,
1810 employer, doctor, etc. This delegation may cover the entire document (an authority level Ref), or
1811 only one or more specific service endpoints within the document (a service level Ref).

1812 Following are the normative rules for processing of the `xrd:Ref` element.

- 1813 1. Ref processing is only be performed if the value of the `refs` media type parameter
1814 (Table 6) is `true` or it is absent or empty. If the value is `false` and the XRD contains at
1815 least one `xrd:Ref` element that could be followed to complete the resolution query, the
1816 resolver MUST return a response with a status code of 262 `REF_NOT_FOLLOWED`. The
1817 rest of the rules below presume that `refs=true`.
- 1818 2. If one or more `xrd:XRD/xrd:Ref` elements are present at the authority level of an XRD,
1819 they MUST be processed automatically before a resolver continues with authority
1820 resolution, performs service endpoint selection, or returns its final output. This rule
1821 applies iteratively to all XRDS documents resolved as a result of Ref processing.
- 1822 3. If an `xrd:XRD/xrd:Ref` element is not present at the authority level of an XRD, but one
1823 or more `xrd:XRD/xrd:Service/xrd:Ref` elements are present in the highest priority
1824 selected service endpoint, they MUST be processed before a resolver completes service
1825 endpoint selection. If all `xrd:XRD/xrd:Service/xrd:Ref` elements fail in the highest
1826 priority selected service endpoint, the resolver MUST try the highest priority
1827 `xrd:XRD/xrd:Service/xrd:Ref` element in the next highest priority selected service
1828 endpoint, and continue until all priority `xrd:XRD/xrd:Service/xrd:Ref` elements in
1829 all selected service endpoints are exhausted. This rule applies iteratively to all XRDS
1830 documents resolved as a result of Ref processing.
- 1831 4. To process a Ref at either the authority or service level, the resolver MUST begin by
1832 selecting the highest priority `xrd:XRD/xrd:Ref` element. The value of the selected
1833 `xrd:Ref` element MUST be a valid absolute XRI. If not, the resolver MUST select the
1834 next highest priority `xrd:Ref` element. If all instances of this element fail, the resolver
1835 MUST return the error 261 `INVALID_REF` in the XRD containing the Ref as defined in
1836 section 14.
- 1837 5. Once a valid `xrd:XRD/xrd:Ref` value is selected, the resolver MUST begin resolution
1838 of a new XRDS document from this XRI using the protocols defined in this specification.
1839 Other than the QXRI, the resolver MUST use the same resolution query parameters as in
1840 the original query. If the Resolution Output Format is an XRDS document, the resolver

- 1841 MUST embed a nested XRDS document containing an XRD representing the Ref as
1842 defined in section 11.4. This XRDS document MUST include a `ref` attribute whose value
1843 is the fully-constructed HTTP(S) URI specified above. This rule applies iteratively to all
1844 XRDS documents resolved as a result of Ref processing.
- 1845 6. If resolution of an `xrd:Ref` element fails at any point in the resolution chain, or if it
1846 succeeds but the resulting XRD does not contain the service endpoint(s) necessary to
1847 continue or complete the original resolution query, the resolver MUST record the nested
1848 XRDS document as far as resolution was successful, including the relevant status codes
1849 for each XRD as specified in section 14. The resolver MUST then select the next highest
1850 priority `xrd:Ref` element in the XRD containing the original `xrd:Ref` element and
1851 repeat step 4. For more details, see section 11.5, *Recursion and Backtracking*.
- 1852 7. If resolution of all `xrd:Ref` elements in the XRD originating Ref processing fails, the
1853 resolver MUST stop and return the error in the requested Resolution Output Format as
1854 defined in section 7.2.
- 1855 8. If resolution of an `xrd:Ref` element succeeds and `cid=true`, the resolver MUST
1856 perform CanonicalID verification across all XRDs in the nested XRDS document as
1857 specified in section 13.2. Note that each set of XRDs in each new nested XRDS
1858 document produced as a result of Redirect or Ref processing constitutes its own
1859 CanonicalID verification chain. *CanonicalID verification never crosses between XRDS*
1860 *documents*.
- 1861 9. If resolution of an `xrd:Ref` element succeeds and the final XRD contains the service
1862 endpoint(s) necessary to continue or complete the original resolution query, further
1863 authority resolution or service endpoint selection MUST continue based on the final XRD
1864 in the nested XRDS document.

1865 11.4 Nested XRDS Documents

1866 If Redirect or Ref reference is successful, it produces a new XRDS document that fully describes
1867 the Redirect or Ref that was followed. If the final requested Resolution Output Format is NOT an
1868 XRDS document, this XRDS document is only needed to obtain the metadata necessary to
1869 continue or complete resolution. However, if the final requested Resolution Output Format is an
1870 XRDS document, this XRDS document produced as a result of Redirect or Ref processing MUST
1871 be included in the containing XRDS document immediately following the `xrd:XRD` element
1872 containing the `xrd:Redirect` or `xrd:Ref` element being followed. This XRDS document is a
1873 recursive authority resolution call and MUST conform to all authority resolution requirements. In
1874 addition, the following rules apply:

- 1875 • For a Redirect, the `xrds:XRDS/@redirect` attribute of the nested XRDS document MUST
1876 contain the fully-constructed HTTP(S) URI it describes as specified in section 11.2.
- 1877 • For a Ref, the `xrds:XRDS/@ref` attribute of the nested XRDS document MUST be contain
1878 the exact value of the `xrd:XRD/xrd:Ref` element it describes.

1879 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
1880 original query identifier even if resolution traverses multiple Redirects or Refs. Note that nested
1881 XRDS documents MUST NOT include an XRD for the community root subsegment because this
1882 is part of the configuration of the resolver.

1883 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD with an
1884 `xml:id` attribute value matching the `xml:id` attribute value of any previous XRD in the chain of
1885 resolution requests beginning with the original QXRI, the resolver MUST replace this XRD with an
1886 empty XRD element. The resolver MUST set this empty element's `xml:idref` attribute value to
1887 the value of the `xrd:XRD/xml:id` attribute of the matched XRD element. This prevents
1888 conflicting `xrd:XRD/xml:id` values.

1889 11.4.1 Redirect Examples

1890 Example #1:

1891 In this example the original query identifier is `xri://@a`. The first XRD contains an authority
1892 Redirect to `http://a.example.com/`. The elements and attributes specific to Redirect
1893 processing are shown in **bold**. CanonicalIDs are included to illustrate the synonym verification
1894 rule.

```
1895 <XRDS xmlns="xri://$xrds" ref="xri://@a">  
1896   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">  
1897     <Query>*a</Query>  
1898     <ProviderID>xri://@</ProviderID>  
1899     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1  
1900     <Redirect>http://a.example.com/</Redirect>  
1901     ...  
1902   </XRD>  
1903   <XRDS redirect="http://a.example.com/">  
1904     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">  
1905       <ProviderID>xri://@</ProviderID>  
1906       <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1  
1907     CID #1  
1908       ...  
1909       <Service>  
1910         <Type>http://openid.net/signon/1.0</Type>  
1911         <URI>http://openid.example.com/</URI>  
1912       </Service>  
1913     </XRD>  
1914   </XRDS>  
1915 </XRDS>
```

1916 Example #2:

1917 In this example the original query identifier is `xri://@a*b*c`. The second XRD contains a
1918 authority resolution service Redirect to `http://other.example.com/`. Note that because
1919 authority resolution is not complete when this Redirect is encountered, it continues in the outer
1920 XRDS after the nested XRDS representing the Redirect. Again, CanonicalIDs are included to
1921 illustrate the synonym verification rule.

```
1922 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">  
1923   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">  
1924     <Query>*a</Query>  
1925     <ProviderID>xri://@</ProviderID>  
1926     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1  
1927     ...  
1928     <Service>  
1929       <Type>xri://$res*auth*($v*2.0)</Type>  
1930       <URI>http://a.example.com/</URI>  
1931     </Service>  
1932   </XRD>  
1933   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">  
1934     <Query>*b</Query>  
1935     <ProviderID>xri://@!1</ProviderID>  
1936     <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2  
1937     ...  
1938     <Service>  
1939       <Type>xri://$res*auth*($v*2.0)</Type>  
1940       <Redirect>http://other.example.com/</Redirect>  
1941     </Service>  
1942   </XRD>  
1943 <XRDS redirect="http://other.example.com/">
```

1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966

```
<XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
  <Query>*b</Query>
  <ProviderID>xri://@!1</ProviderID>
  <CanonicalID>xri://@!1!2</CanonicalID> ;SAME AS XRDS
#1 CID #2
  ...
  <Service>
    <Type>xri://$res*auth*($v*2.0)</Type>
    <URI>http://b.example.com/</URI>
  </Service>
</XRD>
</XRDS>
<XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
  <Query>*c</Query>
  <ProviderID>xri://@!1!2</ProviderID>
  <CanonicalID>xri://@!1!2!3</CanonicalID> ;XRDS #1 CID #3
  ...
  <Service>
    ...final service endpoints described here...
  </Service>
</XRD>
</XRDS>
```

1967 **Example #3:**

1968 In this example the original query identifier is again xri://@a*b*c. This time the final XRD
1969 contains a service Redirect to http://other.example.com/. Because authority resolution is
1970 complete, the outer XRDS ends with a nested XRD representing the service Redirect.

1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002

```
<XRDS xmlns="xri://$xrd*" ref="xri://@a*b*c">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*a</Query>
    <ProviderID>xri://@</ProviderID>
    <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
    ...
    <Service>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <URI>http://a.example.com/</URI>
    </Service>
  </XRD>
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*b</Query>
    <ProviderID>xri://@!1</ProviderID>
    <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2
    ...
    <Service>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <URI>http://b.example.com/</URI>
    </Service>
  </XRD>
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*c</Query>
    <ProviderID>xri://@!1!2</ProviderID>
    <CanonicalID>xri://@!1!2!3</CanonicalID> ;XRDS #1 CID #3
    ...
    <Service>
      <Type>http://openid.net/signon/1.0</Type>
      <Redirect>http://r.example.com/openid</Redirect>
    </Service>
  </XRD>
</XRDS redirect="http://r.example.com/openid">
```

2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014

```
<XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
  <ProviderID>xri://!1!2</ProviderID>
  <CanonicalID>xri://!1!2!3</CanonicalID>           ;SAME AS
XRDS #1 CID #3
  ...
  <Service>
    <Type>http://openid.net/signon/1.0</Type>
    <URI>http://openid.example.com/</URI>
  </Service>
</XRD>
</XRDS>
</XRDS>
```

2015

11.4.2 Ref Examples

2016

Example #1:

2017

In this example the original query identifier is `xri://@a`. The first XRD contains an authority Ref

2018

to `xri://@x*y`.

2019

```
<XRDS xmlns="xri://$xrds" ref="xri://@a">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*a</Query>
    <ProviderID>xri://@</ProviderID>
    <CanonicalID>xri://!1</CanonicalID>           ;XRDS #1 CID #1
    <Ref>xri://@x*y</Ref>
  </XRD>
  <XRDS ref="xri://@x*y">
    <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
      <Query>*x</Query>
      <ProviderID>xri://@</ProviderID>
      <CanonicalID>xri://!7</CanonicalID>         ;XRDS #2 CID #1
      ...
      <Service>
        <Type>xri://$res*auth*($v*2.0)</Type>
        <URI>http://x.example.com/</URI>
      </Service>
    </XRD>
    <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
      <Query>*y</Query>
      <ProviderID>xri://!7</ProviderID>
      <CanonicalID>xri://!7!8</CanonicalID>       ;XRDS #2 CID #2
      ...
      <Service>
        <Type>xri://$res*auth*($v*2.0)</Type>
        <URI>http://y.example.com/</URI>
      </Service>
      <Service>
        <Type>http://openid.net/signon/1.0</Type>
        <URI>http://openid.example.com/</URI>
      </Service>
    </XRD>
  </XRDS>
</XRDS>
```

2053

Example #2:

2054

In this example the original query identifier is `xri://@a*b*c`. The second XRD contains a authority resolution service Ref to `xri://@x*y`. Note that because authority resolution is not complete when this Ref is encountered, it continues in the outer XRDS after the nested XRDS representing the Ref. *Note especially how the CanonicalIDs progress to satisfy the CanonicalID verification rules specified in section 13.2.*

2055

2056

2057

2058

```

2059 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2060   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2061     <Query>*a</Query>
2062     <ProviderID>xri://@</ProviderID>
2063     <CanonicalID>xri://!1</CanonicalID>      ;XRDS #1 CID #1
2064     ...
2065     <Service>
2066       <Type>xri://$res*auth*($v*2.0)</Type>
2067       <URI>http://a.example.com/</URI>
2068     </Service>
2069   </XRD>
2070   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2071     <Query>*b</Query>
2072     <ProviderID>xri://!1</ProviderID>
2073     <CanonicalID>xri://!1!2</CanonicalID>    ;XRDS #1 CID #2
2074     ...
2075     <Service>
2076       <Type>xri://$res*auth*($v*2.0)</Type>
2077       <Ref>xri://@x*y</Ref>
2078     </Service>
2079   </XRD>
2080   <XRDS ref="xri://@x*y">
2081     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2082       <Query>*x</Query>
2083       <ProviderID>xri://@</ProviderID>
2084       <CanonicalID>xri://!7</CanonicalID>    ;XRDS #2 CID #1
2085       ...
2086       <Service>
2087         <Type>xri://$res*auth*($v*2.0)</Type>
2088         <URI>http://x.example.com/</URI>
2089       </Service>
2090     </XRD>
2091     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2092       <Query>*y</Query>
2093       <ProviderID>xri://!7</ProviderID>
2094       <CanonicalID>xri://!7!8</CanonicalID>  ;XRDS #2 CID #2
2095       ...
2096       <Service>
2097         <Type>xri://$res*auth*($v*2.0)</Type>
2098         <URI>http://y.example.com/</URI>
2099       </Service>
2100     </XRD>
2101   </XRDS>
2102   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2103     <Query>*c</Query>
2104     <ProviderID>xri://!1!2</ProviderID>
2105     <CanonicalID>xri://!1!2!3</CanonicalID>  ;XRDS #1 CID #3
2106   IS CHILD OF XRDS #1 CID #2
2107     ...
2108     <Service>
2109       ...final service endpoints described here...
2110     </Service>
2111   </XRD>
2112 </XRDS>

```

2113 **Example #3:**

2114 In this example the original query identifier is again `xri://@a*b*c`. This time the final XRD
2115 contains a service Ref to `xri://@x*y`. Because authority resolution is complete, the outer
2116 XRDS ends with a nested XRDS representing the service Ref.

2117 `<XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">`

```

2118 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2119   <Query>*a</Query>
2120   <ProviderID>xri://@/</ProviderID>
2121   <CanonicalID>xri://@!1</CanonicalID>      ;XRDS #1 CID #1
2122   ...
2123   <Service>
2124     <Type>xri://$res*auth*($v*2.0)</Type>
2125     <URI>http://a.example.com/</URI>
2126   </Service>
2127 </XRD>
2128 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2129   <Query>*b</Query>
2130   <ProviderID>xri://@!1</ProviderID>
2131   <CanonicalID>xri://@!1!2</CanonicalID>    ;XRDS #1 CID #2
2132   ...
2133   <Service>
2134     <Type>xri://$res*auth*($v*2.0)</Type>
2135     <URI>http://a.example.com/</URI>
2136   </Service>
2137 </XRD>
2138 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2139   <Query>*c</Query>
2140   <ProviderID>xri://@!1!2</ProviderID>
2141   <CanonicalID>xri://@!1!2!3</CanonicalID>  ;XRDS #1 CID #3
2142   ...
2143   <Service>
2144     <Type>http://openid.net/signon/1.0</Type>
2145     <Ref>xri://@x*y</Ref>
2146   </Service>
2147 </XRD>
2148 <XRDS ref="xri://@x*y">
2149   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2150     <Query>*x</Query>
2151     <ProviderID>xri://@/</ProviderID>
2152     <CanonicalID>xri://@!7</CanonicalID>    ;XRDS #2 CID #1
2153     ...
2154     <Service>
2155       <Type>xri://$res*auth*($v*2.0)</Type>
2156       <URI>http://x.example.com/</URI>
2157     </Service>
2158   </XRD>
2159   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2160     <Query>*y</Query>
2161     <ProviderID>xri://@!7</ProviderID>
2162     <CanonicalID>xri://@!7!8</CanonicalID>  ;XRDS #2 CID #2
2163     ...
2164     <Service>
2165       <Type>xri://$res*auth*($v*2.0)</Type>
2166       <URI>http://y.example.com/</URI>
2167     </Service>
2168     <Service>
2169       <Type>http://openid.net/signon/1.0</Type>
2170       <URI>http://openid.example.com/</URI>
2171     </Service>
2172   </XRD>
2173 </XRDS>
2174 </XRDS>

```

2175 11.5 Recursion and Backtracking

2176 Both Redirect and Ref processing requires recursive calls to authority resolution that produce
2177 nested XRDS documents. This recursion can continue to any depth, i.e., a Redirect may contain

2178 another Redirect or a Ref, and a Ref may contain another Ref or a Redirect. To avoid confusion,
2179 either in resolvers implementations or in XRDS documents, it is important to clarify how
2180 "backtracking" works.

2181 • When a resolver first encounters a Redirect or a Ref, this is called the *First Recursion*. As
2182 specified in sections 11.2 and 11.3, the resolver is obligated to resolve the highest priority
2183 Redirect or Ref to see if it can satisfy the resolution query. If EITHER resolution of the
2184 Redirect or Ref fails OR if resolution is successful but the resolver is ultimately unable to
2185 continue or complete the original resolution query, the resolver MUST try the next highest
2186 priority Redirect or Ref. The resolver MUST continue until either it is successful or all
2187 Redirects or Refs have failed.

2188 • If a Redirect or Ref leads to another Redirect or Ref, this is called the *Next Recursion*. The
2189 same rules apply to the Next Recursion as apply to the First Recursion.

2190 • If any Next Recursion completely fails, the resolver MUST return to the previous recursion
2191 and continue trying any untried Redirects or Refs until either it is successful or all Redirects
2192 or Refs have failed.

2193 • Only if the resolver returns to the First Recursion and all its Redirects or Refs have failed
2194 does the resolver stop and return an error.

2195 To avoid excessive recursion and inefficient resolution responses, it is RECOMMENDED to use
2196 as few Redirects or Refs in a resolution chain as possible.

2197

12 Service Endpoint Selection

2198

At each iteration of authority resolution, a resolver obtains an XRDS document containing an XRD describing the target authority. To continue authority resolution, or if necessary to locate a specific service endpoint even if authority resolution is complete, the resolver processes the XRD to perform the second phase of resolution called *service endpoint selection*. This section specifies the rules for this process.

2200

2201

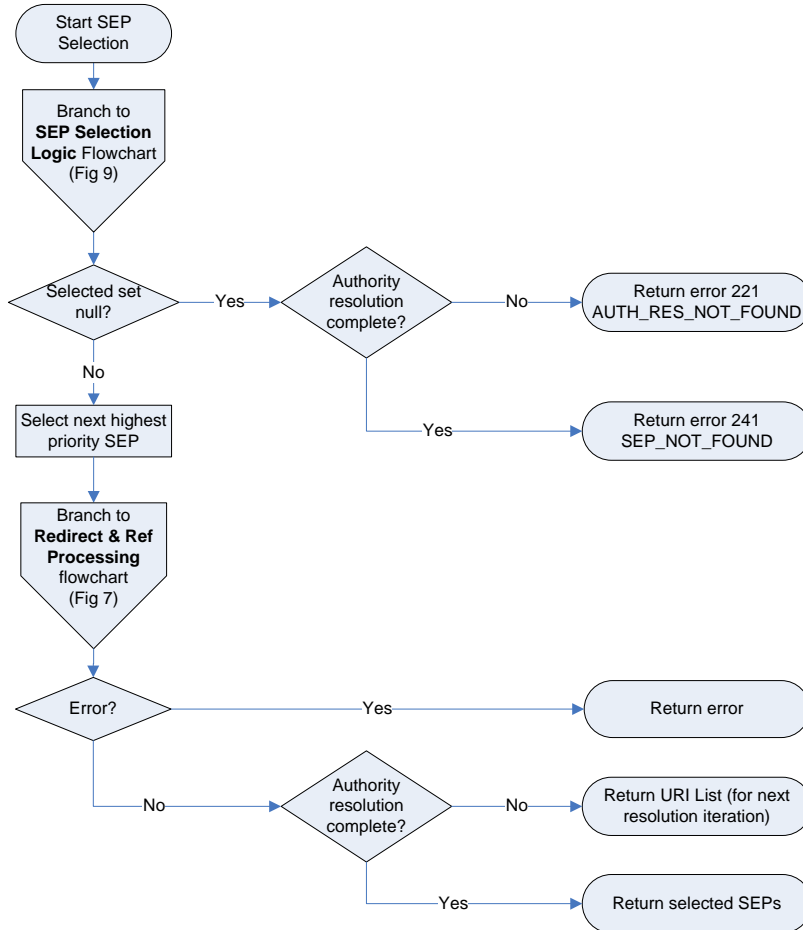
2202

12.1 Processing Rules

2203

Figure 8 (non-normative) shows the overall logical flow of the service endpoint selection process.

2204



2205

2206

Figure 8: Service endpoint selection flowchart.

2207

Following are the normative rules for the overall service endpoint selection process:

2208

1. The inputs for service endpoint selection are defined in Table 8.

- 2209 2. If the final XRD resulting from authority resolution contains an `xrd:XRD/xrd:Redirect`
2210 or `xrd:XRD/xrd:Ref` element, it MUST first be processed as specified in section 11.
- 2211 3. For each `xrd:Service` element in the final XRD, selection MUST follow the service
2212 endpoint selection logic defined in section 12.2. The outcome of this process will be a
2213 selected set of zero or more service endpoints.
- 2214 4. If, after applying the service endpoint selection logic, the selected set is not null and the
2215 highest priority selected service endpoint contains an
2216 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
2217 element, it MUST first be processed as specified in section 11. This is a recursive call
2218 that will produce a nested XRDS document.
- 2219 5. If, after applying the service endpoint selection logic, the selected set is null, the resolver
2220 MUST return the error 241 `AUTH_RES_NOT_FOUND` if authority resolution is not
2221 complete, or 241 `SEP_NOT_FOUND` if authority resolution is complete.
- 2222 6. If authority resolution is complete, the output of service endpoint selection MUST be
2223 returned in a valid Resolution Output Format as defined in Table 10 and conform to the
2224 output requirements defined in section 7.2.

2225

12.2 Service Endpoint Selection Logic

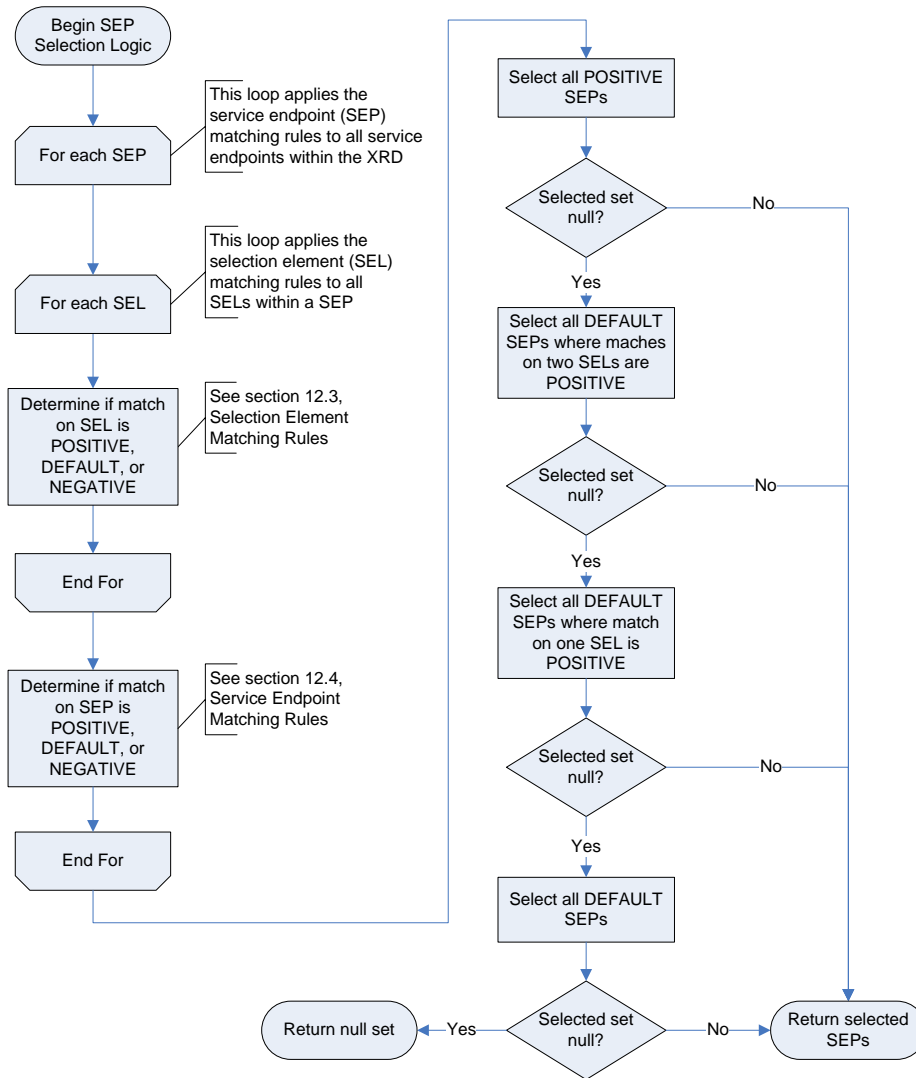
2226

Selection of service endpoints (SEPs) within an XRD is managed using service endpoint selection elements (SEs). As defined in section 4.2.6, `xrd:XRD/xrd:Service` elements may contain three categories of selection elements: `xrd:Type`, `xrd:Path`, and `xrd:MediaType`. Figure 9 (non-normative) shows the overall flow of the service endpoint selection logic.

2227

2228

2229



2230

2231

Figure 9: Service endpoint (SEP) selection logic flowchart.

2232

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

2233

12.3 Selection Element Matching Rules

2234

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

2235 **12.3.1 Selection Element Match Options**

2236 Within each service endpoint, there is a match option for each of the three categories of selection
 2237 elements. The three match options are defined in Table 21:

Match Option	Match Condition	Precedence
POSITIVE	A successful match based on the value of the <code>match</code> attribute as defined in 12.3.2 OR a successful match based the contents of the selection element as defined in sections 12.3.6 - 12.3.8.	1
DEFAULT	The value of the <code>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 12.3.3.	0
NEGATIVE	The selection element does not satisfy either condition above.	-1

2238 *Table 21: Match options for selection elements.*

2239 **12.3.2 The Match Attribute**

2240 All three service endpoint selection elements accept the optional `match` attribute. This attribute
 2241 gives XRDS authors precise control over selection of service endpoints based on the QXRI and
 2242 other resolution input parameters. An enumerated list of the values for this attribute is defined in
 2243 Table 22. If this attribute is present with one of these values, the contents of the selection element
 2244 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.

2245 *Table 22: Enumerated values of the global match attribute and corresponding matching rules.*

2246 *Backwards Compatability Note:* earlier working drafts of this specification included the values
 2247 `match="none"` and `match="contents"`. Both are deprecated. The former is no longer
 2248 supported and the latter is now the default behaviour of any selection element that does not
 2249 include the match attribute or the value of this attribute is empty.

2250 **12.3.3 Absent Selection Element Matching Rule**

2251 If a service endpoint does not contain at least one instance of a particular category of selection
 2252 element, it MUST be considered equivalent to the service endpoint having a DEFAULT match on
 2253 that category of selection element UNLESS the overridden by the `nodefault` parameter as specified
 2254 in Table 22.

2255 12.3.4 Empty Selection Element Matching Rule

2256 If a selection element is present in a service endpoint but the element is empty, and if the element
2257 does not contain a `match` attribute or the value of this attribute is empty, it MUST be considered
2258 equivalent to having a DEFAULT match on this selection element UNLESS the overridden by the
2259 `nodefault` parameter as specified in Table 22.

2260 12.3.5 Multiple Selection Element Matching Rule

2261 Each service endpoint has only one match option for each category of selection element.
2262 Therefore if a service endpoint contains more than one instance of the same category of selection
2263 element (i.e., more than one `xrd:Type`, `xrd:Path`, or `xrd:MediaType` element), matching
2264 MUST be based on the selection element with the highest precedence match option as defined in
2265 Table 21.

2266 12.3.6 Type Element Matching Rules

2267 The following rules apply to matching the value of the input Service Type parameter with the
2268 contents of a non-empty `xrd:XRD/xrd:Service/xrd:Type` element when its `match` attribute
2269 is absent or empty.

- 2270 1. The values of the Service Type parameter and the `xrd:XRD/xrd:Service/xrd:Type`
2271 element SHOULD be normalized according to the requirements of their identifier scheme
2272 prior to input. In particular, if an XRI, IRI, or URI does not include a local part (a path
2273 and/or query component), a trailing forward slash MUST be assumed after the authority
2274 component. In all other cases, a trailing forward slash MUST NOT be assumed, and
2275 therefore is significant in comparisons. In addition, if the value is an XRI or an IRI it
2276 MUST be in URI-normal form as defined in section 4.4. XRI resolvers MAY perform
2277 normalization of these values but MUST NOT be required to do so. As a best practice,
2278 service architects SHOULD assign identifiers for service types that are easy to match and
2279 do not require normalization.
- 2280 2. To result in a POSITIVE match, the values MUST be equivalent according to the
2281 equivalence rules of the applicable identifier scheme. Any other result is a NEGATIVE
2282 match.

2283 12.3.7 Path Element Matching Rules

2284 The following rules apply to matching the value of the input Path String with the contents of a
2285 non-empty `xrd:XRD/xrd:Service/xrd:Path` element when its `match` attribute is absent or
2286 empty.

- 2287 1. If the value is a relative XRI or an IRI it MUST be in URI-normal form as defined in
2288 section 4.4.
- 2289 2. The QXRI Path String being matched MUST NOT include the forward slash separating
2290 an XRI authority segment from the path. Any subsequent forward slash, including trailing
2291 forward slashes, MUST be significant in comparisons.
- 2292 3. Equivalence comparison MUST be performed using Caseless Matching as defined in
2293 section 3.13 of **[Unicode]**, with the exception that it is not necessary to perform
2294 normalization after case folding.
- 2295 4. To result in a POSITIVE match, the value of the Path String (the path portion of the QXRI
2296 as defined in section 7.1.1) MUST be a *subsegment stem match* with the contents of the
2297 `xrd:XRD/xrd:Service/xrd:Path` element. A subsegment stem match is defined as
2298 the entire Path String being character-for-character equivalent with any continuous
2299 sequence of subsegments or segments (including empty subsegments and empty
2300 segments) in the contents of the Path element beginning from the most significant

2301
2302

(leftmost) subsegment. Subsegments and segments are formally defined in **[XRISyntax]**.
Any other result MUST be a NEGATIVE match.

2303

Examples of this rule are shown in Table 23.

Comment [DSR23]: This table is new in ED06.

QXRI (Path in bold)	XRD Path Element	Match
@example	<Path></Path>	POSITIVE
@example	<Path match="null"/>	POSITIVE
@example	<Path>/</Path>	NEGATIVE
@example/	<Path></Path>	POSITIVE
@example/	<Path>/</Path>	NEGATIVE
@example//	<Path></Path>	NEGATIVE
@example//	<Path>/</Path>	POSITIVE
@example//	<Path>/foo</Path>	POSITIVE
@example/foo	<Path>foo</Path>	POSITIVE
@example/foo	<Path>/foo</Path>	NEGATIVE
@example//foo	<Path>foo</Path>	NEGATIVE
@example//foo	<Path>/foo</Path>	POSITIVE
@example/foo*bar	<Path>foo</Path>	NEGATIVE
@example/foo*bar	<Path>foo*bar</Path>	POSITIVE
@example/foo*bar	<Path>foo*bar</Path>	POSITIVE
@example/foo*bar	<Path>foo*bar/baz</Path>	POSITIVE
@example/foo*bar	<Path>foo*bar*baz</Path>	POSITIVE
@example/foo*bar	<Path>foo*bar!baz</Path>	POSITIVE
@example/foo*bar/	<Path>foo*bar</Path>	NEGATIVE
@example/foo*bar/	<Path>foo*bar</Path>	POSITIVE
@example/foo*bar/	<Path>foo*bar/baz</Path>	POSITIVE
@example/foo*bar/	<Path>foo*bar*baz</Path>	NEGATIVE
@example/foo!bar	<Path>foo*bar</Path>	NEGATIVE
@example/foo!bar	<Path>foo!bar*baz</Path>	POSITIVE
@example/(+foo)	<Path>(+foo)</Path>	POSITIVE
@example/(+foo)*bar	<Path>(+foo)</Path>	NEGATIVE
@example/(+foo)*bar	<Path>(+foo) *bar</Path>	POSITIVE
@example/(+foo)*bar	<Path>(+foo) *bar *baz</Path>	POSITIVE
@example/(+foo)!bar	<Path>(+foo) *bar</Path>	NEGATIVE

2304
2305

Table 23: Examples of applying the Path element matching rules.

2306 12.3.8 MediaType Element Matching Rules

2307 The following rules apply to matching the value of the input Service Media Type parameter with
2308 the contents of a non-empty `xrd:XRD/xrd:Service/xrd:MediaType` element when its
2309 `match` attribute is absent or empty.

- 2310 1. The values of the Service Media Type parameter and the `xrd:MediaType` element
2311 SHOULD be normalized according to the rules for media types in section 3.7 of
2312 [RFC2616] prior to input. (The rules are that type and subtype names are case-
2313 insensitive, but parameter values may or may not be case-sensitive depending on the
2314 semantics of the parameter name. XRI Resolution Output Format parameters are case-
2315 insensitive.) XRI resolvers MAY perform normalization of these values but MUST NOT be
2316 required to do so.
- 2317 2. To be a POSITIVE match, the values MUST be character-for-character equivalent. Any
2318 other result is a NEGATIVE match.

2319 12.4 Service Endpoint Matching Rules

2320 The next set of matching rules govern the matching of service endpoints based on the matches of
2321 the selection elements they contain.

2322 12.4.1 Service Endpoint Match Options

2323 For each service endpoint in an XRD, there are three match options as defined in Table 24:

Match Option	Condition
POSITIVE	Meets the Select Attribute Match Rule (section 12.4.2) or the All Positive Match Rule (section 12.4.3).
DEFAULT	Meets the Default Match Rule (section 12.4.4).
NEGATIVE	The service endpoint does not satisfy either condition above.

2324 Table 24: Match options for service endpoints.

2325 12.4.2 Select Attribute Match Rule

2326 All three service endpoint selection elements accept the optional `select` attribute. This attribute
2327 is a Boolean value used to govern matching of the containing service endpoint according to the
2328 following rule. If service endpoint contains a selection element with a POSITIVE match as defined
2329 in section 12.3, and the value of this selection element's `select` attribute is `true`, the service
2330 endpoint automatically MUST be a POSITIVE match, i.e., all other selection elements for this
2331 service endpoint MUST be ignored.

2332 12.4.3 All Positive Match Rule

2333 If a service endpoint has a POSITIVE match on all three categories of selection elements
2334 (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) as defined in section 12.3, the service endpoint
2335 MUST be a POSITIVE match. If even one of the three selection element match types is not
2336 POSITIVE, this rule fails.

2337 12.4.4 Default Match Rule

2338 If a service endpoint fails the Select Attribute Match Rule and the All Positive Match Rule, but
2339 none of the three categories of selection elements has a NEGATIVE match as defined in section
2340 12.3, the service endpoint MUST be a DEFAULT match.

2341 **12.5 Service Endpoint Selection Rules**

2342 The final set of rules governs the selection of service endpoints based on their matches.

2343 **12.5.1 Positive Match Rule**

2344 After applying the matching rules to service endpoints in section 12.4, all service endpoints that
2345 have a POSITIVE match MUST be selected. Only if there are no service endpoints with a
2346 POSITIVE match is the Default Match Rule invoked.

2347 **12.5.2 Default Match Rule**

2348 If the Positive Match Rule above fails, then the service endpoints with a DEFAULT match that
2349 have the highest number of POSITIVE matches on each category of selection element MUST be
2350 selected. This means:

- 2351 1. The service endpoints in the DEFAULT set that have two POSITIVE selection element
2352 matches MUST be selected.
- 2353 2. If the previous set is empty, the service endpoints in the DEFAULT set that have one
2354 POSITIVE selection element match MUST be selected.
- 2355 3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.
- 2356 4. If the previous set is empty, no service endpoint is selected and the return set is null.

2357 **12.6 Pseudocode**

2358 The following pseudocode provides a precise description of the service endpoint selection logic.
2359 The pseudocode is normative, however if there is a conflict between it and the rules stated in the
2360 preceding sections, the preceding sections shall prevail.

2361 The pseudocode uses nine Boolean flags to record the match state for each category of selection
2362 element (SEL) in a service endpoint (SEP):

- 2363 • Postive.Type
2364 • Postive.Path
2365 • Positive.Mediatype
2366 • Default.Type
2367 • Default.Path
2368 • Default.Mediatype
2369 • Matched.Type
2370 • Matched.Path
2371 • Matched.Mediaype

2372 Note that the complete set of nine SEL match flags is needed for each SEP. The pseudocode first
2373 does a loop through all SEPs in the XRD to:

- 2374 1. Set the SEL match flags according to the rules specified in section 12.3;
2375 2. Process the SEL match flags to apply the SEP matching rules specified in section 12.4;
2376 3. Apply the positive SEP selection rule specified in section 12.5.1.

2377 After this loop is complete, the pseudocode tests to see if default SEP selection processing is
2378 required. If so, it performs a second loop applying the default SEP selection rules specified in
2379 section 12.5.2.

2380
2381

FOR EACH SEP

```

2382 CREATE set of SEL match flags
2383 SET all flags to FALSE
2384 FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
2385     SET Matched.x=TRUE
2386     IF match on this SEL is POSITIVE
2387         IF select="true" ;see 12.4.2
2388             ADD SEP TO SELECTED SET
2389             NEXT SEP
2390         ELSE
2391             SET Positive.x=TRUE
2392             NEXT SEL
2393         ENDIF
2394     ELSEIF match on this SEL is DEFAULT ;see 10.3.2 & 12.3.4
2395         IF Positive.x=TRUE ;see 12.3.5
2396             NEXT SEL
2397         ELSEIF nodefault="x" ;see 10.3.2
2398             NEXT SEL
2399         ELSE
2400             SET Default.x=TRUE
2401             NEXT SEL
2402         ENDIF
2403     ELSEIF match on this SEL is NEGATIVE ;see 12.3.1
2404         NEXT SEL
2405     ENDIF
2406 ENDFOR
2407 IF Matched.x=FALSE ;see 12.3.3
2408     IF nodefault_x != TRUE ;see 10.3.2
2409         SET Default.x=TRUE
2410     ENDIF
2411 ENDIF
2412 IF Positive.Type=TRUE AND
2413     Positive.Path=TRUE AND
2414     Positive.Mediatype=TRUE ;see 12.4.3
2415     ADD SEP TO SELECTED SET
2416     NEXT SEP
2417 ELSEIF SELECTED SET != EMPTY ;see 12.5.1
2418     NEXT SEP
2419 ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
2420     (Positive.Path=TRUE OR Default.Path=TRUE) AND
2421     (Positive.MediaType=TRUE OR Default.MediaType=TRUE)
2422     ADD SEP TO DEFAULT SET ;see 12.4.4
2423 ENDIF
2424 ENDFOR
2425 IF SELECTED SET != EMPTY ;see 12.5.1
2426     RETURN SELECTED SET
2427 ENDIF
2428 IF DEFAULT SET != EMPTY ;see 12.5.2
2429     FOR EACH SEP IN DEFAULT SET
2430         IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
2431             (Positive.Type=TRUE AND Positive.MediaType=TRUE) OR
2432             (Positive.Path=TRUE AND Positive.MediaType=TRUE)
2433             ADD SEP TO SELECTED SET
2434         ENDIF
2435     ENDFOR
2436 IF SELECTED SET != EMPTY
2437     RETURN SELECTED SET
2438 ENDIF
2439 FOR EACH SEP IN DEFAULT SET
2440     IF Positive.Type=TRUE OR
2441         Positive.Path=TRUE OR
2442         Positive.MediaType=TRUE
2443         ADD SEP TO SELECTED SET

```

2444
2445
2446
2447
2448
2449
2450
2451
2452

```
                ENDIF
      ENDFOR
      IF SELECTED SET != EMPTY
2447         RETURN SELECTED SET
2448     ELSE
2449         RETURN DEFAULT SET
      ENDIF
2451 ENDIF
2452 RETURN EMPTY SET
```

2453 12.7 Construction of Service Endpoint URIs

2454 The final optional step in the service endpoint selection process is construction of the service
2455 endpoint URI(s). This step is necessary if either:

- 2456 • Automatic URI construction is requested using the `uric` parameter.
- 2457 • The resolution output format is a URI List.

2458 12.7.1 The `uric` Parameter

Comment [DSR24]: This section is new in ED06.

2459 The `uric` media type parameter of the Resolution Output Format is used to govern whether a
2460 resolver should perform construction of the URI automatically for a consuming application.
2461 Following are the processing rules for this parameter:

- 2462 1. If `uric=true`, a resolver MUST apply the URI construction rules specified in section
2463 12.7.2 to each `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD in the
2464 resolution chain. Note that this step is identical to the processing a resolver must perform
2465 to output a URI list.
- 2466 2. The resolver MUST replace the value of each `xrd:XRD/xrd:Service/xrd:URI`
2467 element in the final XRD with the fully constructed URI value.
- 2468 3. The resolver MUST subsequently remove the `append` attribute from each
2469 `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD.
- 2470 4. If `uric=false` or the parameter is absent or empty, a resolver MUST NOT perform any
2471 of the processing specified in this section.

2472 12.7.2 The `Append` Attribute

2473 The `append` attribute of a `xrd:XRD/xrd:Service/xrd:URI` element is used to specify how
2474 the final URI is constructed. The values of this attribute are shown in Table 25.

Value	Component of QXRI to Append
none	None. This is the default if the <code>append</code> attribute is absent or its value is empty
local	The entire local part of the QXRI, defined as being 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

2475 Table 25: Values of the `append` attribute and the corresponding QXRI component to append.

2476 If the `append` attribute is absent or its value is empty, the default value is `none`. Following are the
 2477 rules for construction of the final service endpoint URI based on the value of the `append`
 2478 attribute. Note that these rules must be followed exactly in order to give XRDS document authors
 2479 precise control over construction of service endpoint URIs.

- 2480 1. If the value is `none`, the exact contents of the `xrd:URI` element MUST be returned
 2481 directly without any further processing.
- 2482 2. For any other value, the exact value of the QXRI component specified in Table 25,
 2483 including any leading delimiter(s) and without any additional escaping or percent
 2484 encoding MUST be appended directly to the exact contents of the `xrd:URI` element
 2485 including any trailing delimiter(s). If the value of the QXRI component specified in Table
 2486 25 consists of only a leading delimiter, then this value MUST be appended according to
 2487 these rules. If the value of the QXRI component specified in Table 25 is null, then the
 2488 contents of the `xrd:URI` element MUST be returned directly exactly as if the value of the
 2489 `append` attribute was `none`.
- 2490 3. If any query parameters for XRI proxy resolution were added to an existing QXRI query
 2491 component as defined in section 10.3, these query parameters MUST be removed prior
 2492 to performing the `append` operation as also defined in section 10.3. In particular, if after
 2493 removal of these query parameters the QXRI query component consists of only a *string*
 2494 of one or more question marks (the delimiting question mark plus zero or more additional
 2495 question marks) then *exactly one question mark* MUST also be removed. This preserves
 2496 the query component of the original QXRI if it was null or contained only question marks.

2497 **IMPORTANT:** Construction of HTTP(S) URIs for authority resolution service endpoints is defined
 2498 in section 8.1.10. Note that this involves an additional step taken after all URI construction
 2499 processing in this section is complete. In other words, if the URI element of an XRI authority
 2500 resolution service endpoint includes an `append` attribute, the Next Authority Service URI MUST
 2501 be fully constructed according to the algorithm in this section before appending the Next Authority
 2502 String as defined in section 8.1.10.

2503

13 Synonym Verification

2504
2505
2506
2507

As described in section 5, *XRDS Synonyms*, for security purposes a consuming application must be able to verify the binding between the fully-qualified query identifier (the identifier resolved to an XRDS document) and any synonyms asserted in the final XRD. This especially applies to CanonicalID and CanonicalEquivID synonyms as described in section 5.5.

2508
2509
2510
2511
2512

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 13.1. XRI resolvers automatically perform verification of CanonicalID and CanonicalEquivID synonyms unless this function is explicitly turned off as specified in sections 13.2 and 13.2.3.

2513

13.1 EquivID Verification

2514
2515

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

2516
2517
2518
2519
2520
2521
2522
2523
2524

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

2525

Example:

2526
2527

- Fully-Qualified Query Identifier: `http://example.com/user`
- Asserted EquivID: `xri://=!1000.c78d.402a.8824.bf20`

2528

First XRDS (for `http://example.com/user` — simplified for illustration purposes):

2529
2530
2531
2532
2533
2534
2535
2536
2537
2538

```
<XRDS>
  <XRD>
    <EquivID>xri://=!1000.c78d.402a.8824.bf20</EquivID>
    <CanonicalID>http://example.com/user</CanonicalID>
    <Service priority="10">
      ...
    </Service>
    ...
  </XRD>
</XRDS>
```

2539

Second XRDS (for `xri://=!1000.c78d.402a.8824.bf20`):

2540
2541
2542
2543
2544
2545
2546
2547
2548

```
<XRDS>
  <XRD>
    <Query>!1000.c78d.402a.8824.bf20</Query>
    <ProviderID>xri://=</ProviderID>
    <EquivID>http://example.com/user</EquivID>
    <CanonicalID>xri://=!1000.c78d.402a.8824.bf20</CanonicalID>
    <Service priority="10">
      ...
    </Service>
```

2549
2550
2551

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

2552 The XRD in the second XRDS asserts an EquivID backpointer to the CanonicalID of the XRD in
2553 the first XRDS.

2554 13.2 CanonicalID Verification

2555 XRI resolvers automatically perform verification of CanonicalID and CanonicalEquivID synonyms
2556 unless this function is explicitly turned off using the Resolution Output Format media type
2557 parameter `cid`. The following synonym verification MUST be applied by an XRI resolver if
2558 `cid=true` or the parameter is absent or empty, and MUST NOT be applied if `cid=false`.

- 2559 1. If the value of the `xrd:XRD/xrd:CanonicalID` element is an HTTP(S) URI, it MUST
2560 be verified as specified in section 13.2.1.
- 2561 2. If the value of the `xrd:XRD/xrd:CanonicalID` element is an XRI, it MUST be verified
2562 as specified in section 13.2.2.
- 2563 3. If the value of the `xrd:XRD/xrd:CanonicalID` element is any other identifier,
2564 CanonicalID verification fails and the resolver MUST return the CanonicalID verification
2565 status specified in section 13.2.4.
- 2566 4. If CanonicalID verification succeeds but the final XRD in the resolution chain also
2567 contains a `xrd:XRD/xrd:CanonicalEquivID` element, it MUST also be verified as
2568 specified in section 13.2.3, and the resolver MUST return the CanonicalEquivID
2569 verification status as specified in section 13.2.4.
- 2570 5. In all cases, since synonym verification depends on trusting each authority in the
2571 resolution chain, trusted resolution (section 9) SHOULD be used with either `https=true`
2572 and/or `saml=true` to provide additional assurance of the authenticity of the results.

2573 **IMPORTANT:** There is no guarantee that all XRDSs that describe the same target resource will
2574 return the same verified CanonicalID or CanonicalEquivID. Different parent authorities may assert
2575 different CanonicalIDs or CanonicalEquivIDs for the same child authority and all of these may all
2576 be verifiable. In addition, due to Redirect and Ref processing, the verified CanonicalID or
2577 CanonicalEquivID returned for an XRI MAY differ depending on the resolution input parameters.
2578 For example, as described in section 11, a request for a specific service endpoint type may
2579 trigger processing of a Redirect or Ref resulting in a nested XRDS document. The final XRD in
2580 the nested XRDS document may come from a different parent authority and have a different but
2581 still verifiable CanonicalID or CanonicalEquivID.

2582 13.2.1 HTTP(S) URI Verification Rules

2583 To verify that an HTTP(S) URI is a valid CanonicalID synonym for a query identifier, an XRI
2584 resolver MUST verify that the following tests are successful:

- 2585 1. The query identifier MUST be an HTTP(S) URI.
- 2586 2. The query identifier MUST be resolved as specified in section 6.
- 2587 3. The asserted CanonicalID synonym MUST be an HTTP(S) URI equivalent to: a) the
2588 query identifier, or b) the query identifier plus a valid fragment as defined by **[RFC3986]**.

2589 If the `xrd:XRD/xrd:CanonicalID` element contains any other HTTP(S) URI, or any other URI
2590 except an XRI, CanonicalID verification fails.

2591 13.2.2 XRI Verification Rules

2592 To verify that an XRI is a valid CanonicalID synonym for a query identifier, an XRI resolver MUST
2593 verify that all the following tests are successful.

- 2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
1. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:ProviderID` element in the XRD from the community root authority **MUST** match the value of the `xrd:XRD/xrd:CanonicalID` element configured in the XRI resolver or available in a self-describing XRD from the community root authority (or its equivalent). See section 8.1.6.
 2. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID` element **MUST** be a direct child of the value of the `xrd:XRD/xrd:ProviderID` element. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element is `@!1`, then the value of the `xrd:XRD/xrd:ProviderID` element must be `@`.
 3. For each subsequent XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID` element **MUST** be a direct child of the value of the `xrd:XRD/xrd:CanonicalID` element in the parent XRD. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element asserted in an XRD is `@!1!2!3`, then the value of the `xrd:XRD/xrd:CanonicalID` element in the XRD of the parent authority must be `@!1!2`.
 4. If Redirect or Ref processing is required during resolution as specified in section 11, the rules above **MUST** also apply for each nested XRDS document. **IMPORTANT**: each set of XRDs in each new nested XRDS document produced as a result of Redirect or Ref processing constitutes its own CanonicalID verification chain. *CanonicalID verification never crosses between XRDS documents.* See the examples in section 11.4.

2614 13.2.3 CanonicalEquivID Verification

2615 CanonicalID verification also requires verification of a CanonicalEquivID *only if it is present in the*
2616 *final XRD in the resolution chain.* Since CanonicalEquivID verification requires an extra resolution
2617 cycle, restricting automatic verification to the final XRD in the resolution chain ensures it will add
2618 at most one additional resolution cycle.

2619 CanonicalEquivID verification is accomplished by verifying that an EquivID backpointer exists in
2620 the XRD obtained by resolving the CanonicalEquivID value. To verify that either an HTTP(S) URI
2621 or an XRI is a valid hierarchical CanonicalEquivID synonym for a query identifier, an XRI resolver
2622 **MUST** verify that all the following tests are successful:

- 2623
2624
2625
2626
2627
2628
2629
2630
2631
1. CanonicalID verification as specified in section 13.2 **MUST** have completed successfully.
 2. The asserted CanonicalEquivID value **MUST** be a valid HTTP(S) URI or XRI.
 3. The asserted CanonicalEquivID value **MUST** resolve successfully to an XRDS document according to the rules in this specification *using the same resolution parameters as in the original resolution request.*
 4. The final XRD in the XRDS document **MUST** contain a `xrd:XRD/xrd:EquivID` element whose value is equivalent to the value of the verified `xrd:XRD/xrd:CanonicalID` element in the XRD asserting the CanonicalEquivID synonym.

2632 **SPECIAL SECURITY CONSIDERATION:** See section 5.2.2 regarding the rules for provisioning
2633 of an `xrd:XRD/xrd:EquivID` element in an XRD.

2634 13.2.4 Verification Status Attributes

2635 If CanonicalID verification is performed, an XRI resolver **MUST** return the CanonicalID and
2636 CanonicalEquivID status using an attribute of the `xrd:XRD/xrd:Status` element in each XRD
2637 in the output as follows:

- 2638
2639
1. CanonicalID verification **MUST** be reported using the `cid` attribute.
 2. CanonicalEquivID verification **MUST** be reported using the `ceid` attribute.

- 2640 3. Both attributes accept four enumerated values: `absent` if the element is not present, `off`
2641 if verification is not performed, `verified` if the element is verified, and `failed` if
2642 verification fails.
- 2643 4. The `off` value applies to both elements if CanonicalID verification is not performed
2644 (`cid=false`)
- 2645 5. The `off` value applies to the CanonicalEquiVID element in any XRD before the final XRD
2646 if CanonicalID verification is performed (`cid=true`), because a resolver only verifies this
2647 element in the final XRD.

2648 From these attributes, a consuming application knows on every XRD in the XRDS document
2649 whether the CanonicalID is present and has been verified. In addition, for the final XRD in the
2650 XRDS document, it knows whether the CanonicalEquiVID element is present and has been
2651 verified.

2652 13.2.5 Examples

2653 Example #1:

- 2654 • Fully-Qualified Query Identifier: `http://example.com/user`
- 2655 • Asserted CanonicalID: `http://example.com/user#1234`

2656 XRDS (simplified for illustration purposes):

```
2657 <XRDS>  
2658 <XRD>  
2659 <CanonicalID>http://example.com/user#1234</CanonicalID>  
2660 <Service priority="10">  
2661 ...  
2662 </Service>  
2663 ...  
2664 </XRD>  
2665 </XRDS>
```

2666 The asserted CanonicalID satisfies the HTTP(S) URI verification rules in section 13.2.1.

2667

2668 Example #2:

- 2669 • Fully-Qualified Query Identifier: `=example.name*delegate.name`
- 2670 • Asserted CanonicalID: `!=1000.62b1.44fd.2855!1234`

2671 XRDS (for `=example.name*delegate.name`):

```
2672 <XRDS>  
2673 <XRD>  
2674 <Query>*example.name</Query>  
2675 <ProviderID>xri://=</ProviderID>  
2676 <LocalID>!1000.62b1.44fd.2855</LocalID>  
2677 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>  
2678 <Service>  
2679 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>  
2680 <Type>xri://$res*auth*($v*2.0)</Type>  
2681 <MediaType>application/xrds+xml</MediaType>  
2682 <URI priority="10">http://resolve.example.com</URI>  
2683 <URI priority="15">http://resolve2.example.com</URI>  
2684 <URI>https://resolve.example.com</URI>  
2685 </Service>  
2686 ...  
2687 </XRD>
```

```
2688 <XRD>
2689 <Query>*delegate.name</Query>
2690 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2691 <LocalID>!1234</LocalID>
2692 <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
2693 <Service priority="1">
2694 ...
2695 </Service>
2696 ...
2697 </XRD>
2698 </XRDS>
```

2699 The asserted CanonicalID satisfies the XRI verification rules in section 13.2.2.

2700

2701 **Example #3:**

- 2702 • Fully-Qualified Query Identifier: `http://example.com/user`
- 2703 • Asserted CanonicalID: `http://example.com/user`
- 2704 • Asserted CanonicalEquivID: `https://different.example.net/path/user`

2705 First XRDS (for `http://example.com/user`):

```
2706 <XRDS>
2707 <XRD>
2708 <CanonicalID>http://example.com/user</CanonicalID>
2709 <CanonicalEquivID>
2710 https://different.example.net/path/user
2711 </CanonicalEquivID>
2712 <Service priority="10">
2713 ...
2714 </Service>
2715 ...
2716 </XRD>
2717 </XRDS>
```

2718 Second XRDS (for `https://different.example.net/path/user`):

```
2719 <XRDS>
2720 <XRD>
2721 <EquivID>http://example.com/user</EquivID>
2722 <CanonicalID>https://different.example.net/path/user</CanonicalID>
2723 <Service priority="10">
2724 ...
2725 </Service>
2726 ...
2727 </XRD>
2728 </XRDS>
```

2729 The asserted CanonicalEquivID satisfies the verification rules in section 13.2.3 because it
2730 resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of the first
2731 XRDS.

2732

2733 **Example #4:**

- 2734 • Fully-Qualified Query Identifier: `http://example.com/user`
- 2735 • Asserted CanonicalID: `http://example.com/user`
- 2736 • Asserted CanonicalEquivID: `=!1000.62b1.44fd.2855`

2737 XRDS (for `http://example.com/user`):

```
2738 <XRDS>
2739 <XRD>
2740 <CanonicalID>http://example.com/user</CanonicalID>
2741 <CanonicalEquivID>xri://=!1000.62b1.44fd.2855</CanonicalEquivID>
2742 <Service priority="10">
2743   ...
2744 </Service>
2745   ...
2746 </XRD>
2747 </XRDS>
```

2748 XRDS (for `xri://=!1000.62b1.44fd.2855`):

```
2749 <XRDS>
2750 <XRD>
2751 <Query>!1000.62b1.44fd.2855</Query>
2752 <ProviderID>xri://=</ProviderID>
2753 <EquivID>http://example.com/user</EquivID>
2754 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2755 <Service priority="10">
2756   ...
2757 </Service>
2758   ...
2759 </XRD>
2760 </XRDS>
```

2761

2762 **Example #5:**

- 2763 • Fully-Qualified Query Identifier: `=example.name`
- 2764 • Asserted CanonicalID: `xri://=!1000.62b1.44fd.2855`
- 2765 • Asserted CanonicalEquivID: `https://example.com/user`

2766 First XRDS (for `=example.name`):

```
2767 <XRDS>
2768 <XRD>
2769 <Query>*example.name</Query>
2770 <ProviderID>xri://=</ProviderID>
2771 <LocalID>!1000.62b1.44fd.2855</LocalID>
2772 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2773 <CanonicalEquivID>https://example.user</CanonicalEquivID>
2774 <Service priority="10">
2775   ...
2776 </Service>
2777   ...
2778 </XRD>
2779 </XRDS>
```

2780 Second XRDS (for `https://example.com/user`):

```
2781 <XRDS>
2782 <XRD>
2783 <EquivID>xri://=!1000.62b1.44fd.2855</EquivID>
2784 <CanonicalID>https://example.com/user</CanonicalID>
2785 <Service priority="10">
2786   ...
2787 </Service>
2788   ...
```

2789 </XRD>
2790 </XRDS>

2791

2792 **Example #6:**

- 2793 • Fully-Qualified Query Identifier: =example.name*delegate.name
- 2794 • Asserted CanonicalID: xri://=!1000.62b1.44fd.2855!1234
- 2795 • Asserted CanonicalEquivID: @!1000.f3da.9056.aca3!5555

2796 First XRDS (for =example.name*delegate.name):

```
2797 <XRDS>
2798 <XRD>
2799 <Query>*example.name</Query>
2800 <ProviderID>xri://=!</ProviderID>
2801 <LocalID>!1000.62b1.44fd.2855</LocalID>
2802 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2803 <Service>
2804 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2805 <Type>xri://$res*auth*($v*2.0)</Type>
2806 <MediaType>application/xrds+xml</MediaType>
2807 <URI priority="10">http://resolve.example.com</URI>
2808 <URI priority="15">http://resolve2.example.com</URI>
2809 <URI>https://resolve.example.com</URI>
2810 </Service>
2811 ...
2812 </XRD>
2813 <XRD>
2814 <Query>*delegate.name</Query>
2815 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2816 <LocalID>!1234</LocalID>
2817 <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
2818 <CanonicalEquivID>
2819 xri://@11000.f3da.9056.aca3!5555
2820 </CanonicalEquivID>
2821 <Service priority="1">
2822 ...
2823 </Service>
2824 ...
2825 </XRD>
2826 </XRDS>
```

2827 • Second XRDS (for @!1000.f3da.9056.aca3!5555):

```
2828 <XRDS>
2829 <XRD>
2830 <Query>!1000.f3da.9056.aca3</Query>
2831 <ProviderID>xri://@</ProviderID>
2832 <CanonicalID>xri://@11000.f3da.9056.aca3</CanonicalID>
2833 <Service>
2834 <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
2835 <Type>xri://$res*auth*($v*2.0)</Type>
2836 <MediaType>application/xrds+xml</MediaType>
2837 <URI priority="10">http://resolve.example.com</URI>
2838 <URI priority="15">http://resolve2.example.com</URI>
2839 <URI>https://resolve.example.com</URI>
2840 </Service>
2841 ...
2842 </XRD>
```

2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854

```
<XRD>  
  <Query>!5555</Query>  
  <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>  
  <LocalID>!5555</LocalID>  
  <EquivID>xri://=!1000.62b1.44fd.2855!1234</EquivID>  
  <CanonicalID>xri://@!1000.f3da.9056.aca3!5555</CanonicalID>  
  <Service priority="1">  
    ...  
  </Service>  
  ...  
</XRD>  
</XRDS>
```

2855

14 Status Codes and Error Processing

2856

14.1 Status Elements

Comment [DSR25]: This section is new in ED06.

2857

XRDS architecture uses two XRD elements for status reporting:

2858

- The `xrd:XRD/xrd:ServerStatus` element is used by an authority resolution service to report the server-side status of a resolution query to a resolver.

2859

2860

- The `xrd:XRD/xrd:Status` element is used by a resolver to report the client-side status of a resolution query to a consuming application. Note that attributes and contents of this element MAY differ from those of the `xrd:XRD/xrd:ServerStatus` element due to either client-side error detection or reporting of CanonicalID verification status (section 13.2.4).

2861

2862

2863

2864

Following are the normative rules that apply to usage of these elements:

2865

1. For XRDS servers and clients, each of these elements is OPTIONAL.

2866

2. An XRI authority resolution service is REQUIRED to include an `xrd:XRD/xrd:ServerStatus` element for each XRD in a resolution response. (See the backwards compatibility note below.)

2867

2868

2869

3. An XRI resolver is REQUIRED to add an `xrd:XRD/xrd:ServerStatus` element to each XRD if the Resolution Output Format is an XRDS or an XRD document.

2870

2871

4. In SAML trusted resolution, the `xrd:XRD/xrd:Status` element MUST be excluded from signature verification as specified in section 9.2.4, as it is added by the resolver after receiving the signed response from the server.

2872

2873

2874

5. These elements MUST include the status codes specified in section 14.2 as the value of the required `code` attribute.

2875

2876

6. These elements SHOULD contain the status context strings specified in section 14.3. Authority resolution services or resolvers MAY add additional information to status context strings.

2877

2878

2879

2880

BACKWARDS COMPATABILITY NOTE: The `xrd:XRD/xrd:ServerStatus` element was not included in earlier versions of this specification. If an older authority resolution server does not produce this element in generic or HTTPS trusted resolution, a resolver SHOULD generate it. For SAML trusted resolution, a resolver MUST NOT generate it.

2881

2882

2883

2884

14.2 Status Codes

2885

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

2886

2887

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

2888

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

2889

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

2890

2891

The 2xx and 3xx categories are broken into seven minor categories:

2892

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

2893

- x1x: Input error

2894

- x2x: Generic authority resolution error.

- 2895 • x3x: Trusted authority resolution error.
- 2896 • x4x: Service endpoint (SEP) selection error.
- 2897 • [x5x: Redirect error.](#)
- 2898 • [x6x: Ref error.](#)

Formatted: Bullets and Numbering
 Formatted: Space After: 12 pt

2899 The full list of XRI resolution status codes is defined in Table 26.
 2900

Code	Symbolic Status	Phase(s)	Description
100	SUCCESS	Any	Operation was successful.
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_OUTPUT_FORMAT	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	The resolver was unable to locate an

Deleted: 101 ... [1]

Deleted: R
 Deleted: ES_MEDIA_TYPE

		resolution	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.
250	REDIRECT_ERROR	Redirect Processing	Generic Redirect error.
251	INVALID_REDIRECT	Redirect Processing	At least one Redirect element was found but resolution failed.
252	INVALID_HTTPS_REDIRECT	Redirect Processing	https=true but a Redirect element containing an HTTPS URI was not found.
253	REDIRECT_VERIFY_FAILED	Redirect Processing	Synonym verification failed in an XRD after following a redirect. See section 11.2
260	REF_ERROR	Ref Processing	Generic Ref processing error.
261	INVALID_REF	Ref Processing	A valid Ref XRI was not found.
262	REF_NOT_FOLLOWED	Ref Processing	At least one Ref was present but the refs parameter was set to false.
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).

Formatted: XMLPath

Formatted: XMLPath

Formatted: XMLPath

2901 *Table 26: Error codes for XRI resolution.*

2902 **14.3 Status Context Strings**

2903 Each status code in Table 26 MAY be returned with an optional status context string that provides
2904 additional human-readable information about the status or error condition. When the Resolution
2905 Output Format is an XRDS Document or XRD Document, this string is returned as the contents of
2906 the `xrd:XRDS/xrd:ServerStatus` and `xrd:XRDS/xrd>Status` elements. When the
2907 Resolution Output Format is a URI List, this string MUST be returned as the second line of a plain
2908 text message as specified in section 10.6. Implementers SHOULD provide error context strings
2909 with additional information about an error and possible solutions whenever it can be helpful to
2910 developers or end users.

2911 **14.4 Returning Errors in Plain Text or HTML**

2912 If the Resolution Output Format is a URI List as defined in section 7.2, an error MUST be
2913 returned with the content type `text/plain`. In this content:

- 2914 • The first line MUST consist of only the numeric error code as defined in section 14.2 followed
2915 by a CRLF.
- 2916 • The second line is OPTIONAL; if present it MUST contain the error context string as defined
2917 in section 14.3.

2918 The same rules apply if the Resolution Output Format is an HTTP(S) Redirect as defined in
2919 section 7.2, except the media type MAY also be `text/html`. It is particularly important in this
2920 case to return an error message that will be understandable to an end-user who may have no
2921 understanding of XRI resolution or the fact that the error is coming from an XRI proxy resolver.

2922 **14.5 Error Handling in Recursing and Proxy Resolution**

2923 In recursing and proxy resolution (sections 8.1.8 and 10), a server is acting as a client resolver for
2924 other authority resolution service endpoints. If in this intermediary capacity it receives an
2925 unrecoverable error, it MUST return the error to the originating client in the output format
2926 specified by the value of the requested Resolution Output Format as defined in section 7.2.

2927 If the output format is an XRDS Document, it MUST contain `xrd:XRDS` elements for all
2928 subsegments successfully resolved or retrieved from cache prior to the error. Each XRD MUST
2929 include the `xrd:ServerStatus` element as reported by the authoritative server. The final
2930 `xrd:XRDS` element MUST include the `xrd:Query` element that produced the error and the
2931 `xrd>Status` element that describes the error as defined above.

2932 If the output format is an XRD Document, it MUST include the `xrd:Query` element that produced
2933 the error, the `xrd:ServerStatus` element as reported by the authoritative server, and the
2934 `xrd>Status` element that describes the error as defined above.

2935 If this output format is a URI List or an HTTP(S) redirect, a proxy resolver SHOULD return a
2936 human-readable error message as specified in section 14.4.

Comment [DSR26]: This section was added new in ED06 (although the normative text was already in the overall section.)

2937 15 Use of HTTP(S)

2938 15.1 HTTP Errors

2939 When a resolver encounters fatal HTTP(S) errors during the resolution process, it **MUST** return
2940 the appropriate XRI resolution error code and error message as defined in section 14. In this way
2941 calling applications do not have to deal separately with XRI and HTTP error messages.

2942 15.2 HTTP Headers

2943 15.2.1 Caching

2944 The HTTP caching capabilities described by **[RFC2616]** should be leveraged for all types of XRI
2945 resolution service. Specifically, implementations **SHOULD** implement the caching model
2946 described in section 13 of **[RFC2616]**, and in particular, the “Expiration Model” of section 13.2, as
2947 this requires the fewest round-trip network connections.

2948 All XRI resolution servers **SHOULD** send the Cache-Control or Expires headers in their
2949 responses per section 13.2 of **[RFC2616]** unless there are overriding security or policy reasons to
2950 omit them.

2951 Note that HTTP Cache headers **SHOULD NOT** conflict with expiration information in an XRD.
2952 That is, the expiration date specified by HTTP caching headers **SHOULD NOT** be later than any
2953 of the expiration dates for any of the `xrd:Expires` elements returned in the HTTP response.
2954 This implies that recursing and proxy resolvers **SHOULD** compute the “soonest” expiration date
2955 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching
2956 headers for the HTTP response.

2957 15.2.2 Location

2958 During HTTP interaction, “Location” headers may be present per **[RFC2616]** (i.e., during 3XX
2959 redirects). Redirects **SHOULD** be made cacheable through appropriate HTTP headers, as
2960 specified in section 15.2.1.

2961 15.2.3 Content-Type

2962 For authority resolution, the “Content-type” header in the 2XX responses **MUST** contain the
2963 media type identifier values specified in Table 11 (for generic resolution), Table 15 (for HTTPS
2964 trusted resolution), Table 16 (for SAML trusted resolution), or Table 17 (or HTTPS+SAML trusted
2965 resolution).

2966 Following service endpoint selection, clients and servers **MAY** negotiate content type using
2967 standard HTTP content negotiation features. Regardless of whether this feature is used,
2968 however, the server **MUST** respond with an appropriate media type in the “Content-type” header
2969 if the resource is found and an appropriate content type is returned.

2970 15.3 Other HTTP Features

2971 HTTP provides a number of other features including transfer-coding, proxying, validation-model
2972 caching, and so forth. All these features may be used insofar as they do not conflict with the
2973 required uses of HTTP described in this document.

2974 15.4 Caching and Efficiency

2975 15.4.1 Resolver Caching

2976 In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the
2977 application level. For best results, however, resolution clients SHOULD be conservative with
2978 caching expiration semantics, including cache expiration dates. This implies that in a series of
2979 HTTP redirects, for example, the results of the entire process SHOULD only be cached as long
2980 as the shortest period of time allowed by any of the intermediate HTTP responses.

2981 Because not all HTTP client libraries expose caching expiration to applications, identifier
2982 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the
2983 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments
2984 should be mindful of limitations in current HTTP clients and proxies.

2985 The cache expiration time of an XRD may also be explicitly limited by the parent authority. If the
2986 expiration time in the `xrd:Expires` element is sooner than the expiration time calculated from
2987 the HTTP caching semantics, the XRD MUST be discarded before the expiration time in
2988 `xrd:Expires`. Note also that a `saml:Assertion` element returned during SAML trusted
2989 resolution has its own signature expiration semantics as defined in [SAML]. While this may
2990 invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if
2991 it is not expired by HTTP caching semantics or the `xrd:Expires` element.

2992 With both application-level and HTTP-level caching, the resolution process is designed to have
2993 minimal overhead. Resolution of each qualified subsegment of an XRI authority segment is a
2994 separate step described by a separate XRD, so intermediate results can typically be cached in
2995 their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified
2996 subsegments, which are common to more identifiers, will naturally result in a greater number of
2997 cache hits than resolution of lower-level subsegments.

2998 15.4.2 Synonyms

2999 The publication of synonyms in XRDS documents (section 5) can further increase cache
3000 efficiency. If an XRI resolution request produces a cache hit on a synonym, the following rules
3001 apply:

- 3002 1. If the cache hit is on a LocalID synonym, the resolver MAY return the cached XRD
3003 document if: a) it is from the correct ProviderID, b) it has not expired, and c) it was
3004 obtained using the same trusted resolution and synonym verification parameters as the
3005 current resolution request.
- 3006 2. If the cache hit is on a CanonicalID synonym, the resolver MAY return the entire cached
3007 XRDS document if: a) it has not expired, and b) it was obtained using the same trusted
3008 resolution and synonym verification parameters as the current resolution request.

3009 IMPORTANT: The effect of these rules is that the application calling an XRI resolver MAY receive
3010 back an XRD document, or an XRDS document containing XRD document(s), in which the
3011 `<Query>` element does not match the resolution request, but in which a `<LocalID>` element
3012 does match the resolution request.

3013 16 Extensibility and Versioning

3014 16.1 Extensibility

3015 16.1.1 Extensibility of XRDs

3016 The XRD schema in Appendix A use an an open-content model that is designed to be extended
3017 with other metadata. In most places, extension elements and attributes from namespaces other
3018 than `xri://$xrd*($v*2.0)` are explicitly allowed. These extension points are designed to
3019 simplify default processing using a “Must Ignore” rule. The base rule is that unrecognized
3020 elements and attributes, and the content and child elements of unrecognized elements, MUST be
3021 ignored. As a consequence, elements that would normally be recognized by a processor MUST
3022 be ignored if they appear as descendants of an unrecognized element.

3023 Extension elements MUST NOT require new interpretation of elements defined in this document.
3024 If an extension element is present, a processor MUST be able to ignore it and still correctly
3025 process the XRD document.

3026 Extension specifications MAY simulate “Must Understand” behavior by applying an “enclosure”
3027 pattern. Elements defined by the XRD schema in Appendix A whose meaning or interpretation is
3028 modified by extension elements can be wrapped in a extension container element defined by the
3029 extension specification. This extension container element SHOULD be in the same namespace
3030 as the other extension elements defined by the extension specification.

3031 Using this design, all elements whose interpretations are modified by the extension will now be
3032 contained in the extension container element and thus will be ignored by clients or other
3033 applications unable to process the extension. The following example illustrates this pattern using
3034 an extension container element from an extension namespace (`other:SuperService`) that
3035 contains an extension element (`other:ExtensionElement`):

```
3036 <XRD>  
3037   <Service>  
3038     ...  
3039   </Service>  
3040   <other:SuperService>  
3041     <Service>  
3042       ...  
3043       <other:ExtensionElement>...</other:ExtensionElement>  
3044     </Service>  
3045   </other:SuperService>  
3046 </XRD>
```

3047 In this example, the `other:ExtensionElement` modifies the interpretation or processing rules
3048 for the parent `xrd:Service` element and therefore must be understood by the consumer for the
3049 proper interpretation of the parent `xrd:Service` element. To preserve the correct interpretation
3050 of the `xrd:Service` element in this context, the `xrd:Service` element is “wrapped” so only
3051 consumers that understand elements in the `other:SuperService` namespace will attempt to
3052 process the `xrd:ProviderID` element.

3053 The addition of extension elements does not change the requirement for SAML signatures to be
3054 verified across all elements, whether recognized or not.

3055 16.1.2 Other Points of Extensibility

3056 The use of HTTP, XML, XRI, and URIs in the design of XRDS documents, XRD elements, and
3057 XRI resolution architecture provides additional specific points of extensibility:

- 3058 • Specification of new resolution service types or other service types using XRI or URIs as
3059 values of the `xrd:Type` element.
- 3060 • Specification of new resolution output formats or features using media types and media type
3061 parameters as values of the `xrd:MediaType` element as defined in [RFC2045] and
3062 [RFC2046].
- 3063 • HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- 3064 • Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- 3065 • Use of cross-references within XRIs, particularly for associating new types of metadata with a
3066 resource. See [XRISyntax] and [XRIMetadata].

3067 16.2 Versioning

3068 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,
3069 this section describes versioning guidelines.

3070 In general, this specification follows the same versioning guidelines as established in section
3071 4.2.1 of [SAML]:

3072 *In general, maintaining namespace stability while adding or changing the content of a*
3073 *schema are competing goals. While certain design strategies can facilitate such changes,*
3074 *it is complex to predict how older implementations will react to any given change, making*
3075 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*
3076 *minor revisions is reserved, in the interest of namespace stability. Except in special*
3077 *circumstances (for example, to correct major deficiencies or to fix errors),*
3078 *implementations should expect forward-compatible schema changes in minor revisions,*
3079 *allowing new messages to validate against older schemas.*

3080 *Implementations SHOULD expect and be prepared to deal with new extensions and*
3081 *message types in accordance with the processing rules laid out for those types. Minor*
3082 *revisions MAY introduce new types that leverage the extension facilities described in [this*
3083 *section]. Older implementations SHOULD reject such extensions gracefully when they*
3084 *are encountered in contexts that dictate mandatory semantics.*

3085 16.2.1 Version Numbering

3086 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number
3087 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version
3088 number $Major_A.Minor_A$ if and only if:

3089 $Major_B > Major_A$ OR ($Major_B = Major_A$) AND $Minor_B > Minor_A$)

3090 16.2.2 Versioning of the XRI Resolution Specification

3091 New releases of the XRI Resolution specification may specify changes to the resolution protocols
3092 and/or the XRD schema in Appendix A. When changes affect either of these, the resolution
3093 service type version number will be changed. Where changes are purely editorial, the version
3094 number will not be changed.

3095 In general, if a change is backward-compatible, the new version will be identified using the
3096 current major version number and a new minor version number. If the change is not backward-
3097 compatible, the new version will be identified with a new major version number.

3098 16.2.3 Versioning of Protocols

3099 The protocols defined in this document may also be versioned by future releases of the XRI
3100 Resolution specification. If these protocols are not backward-compatible with older

3101 implementations, they will be assigned a new XRI with a new version identifier for use in
3102 identifying their service type in XRDs. See section 3.1.2.

3103 Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP
3104 provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an
3105 XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely
3106 to continue to use the same XRI to identify the protocol as was used in previous versions of the
3107 XRI Resolution specification.

3108 **16.2.4 Versioning of XRDs**

3109 The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have
3110 no specific knowledge of the elements it may contain. Therefore it has no version indicator, and
3111 can remain stable indefinitely because there is no need to version its namespace.

3112 The `xrd:XRD` element has a `version` attribute. [This attribute is OPTIONAL for this version of](#)
3113 [the XRI resolution specification \(version 2.0\). This attribute will be REQUIRED for all future](#)
3114 [versions of this specification.](#) When used, the value of this attribute MUST be the exact numeric
3115 version value of the XRI Resolution specification to which its containing elements conform.

3116 When new versions of the XRI Resolution specification are released, the namespace for the XRD
3117 schema may or may not be changed. If there is a major version number change, the namespace
3118 for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the
3119 namespace for the `xrd:XRD` schema may remain unchanged.

3120 [Note that conformance to a specific XRD version does not preclude an author from including](#)
3121 [extension elements from a different namespace in the XRD. See section 16.1 above.](#)

3122

17 Security and Data Protection

3123
3124
3125
3126

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.

3127

17.1 DNS Spoofing or Poisoning

3128
3129
3130
3131
3132
3133
3134
3135
3136

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.

3137

17.2 HTTP Security

3138
3139
3140
3141
3142
3143
3144
3145
3146
3147

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.

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.

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

3148

17.3 SAML Considerations

3149
3150
3151

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.

3152

17.4 Limitations of Trusted Resolution

3153
3154
3155
3156
3157

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.

3158
3159
3160

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.

3161 17.5 Synonym Verification

3162 As discussed in section 5, XRI and XRDS infrastructure has rich support for identifiers synonyms,
3163 including identifier synonyms that cross security domains. For this reason it is particularly
3164 important that identifier authorities, including registries, registrars, directory administrators,
3165 identity brokers, and other parties who issue XRIs and manage XRDS documents, enforce the
3166 security policies highlighted in section 5 regarding registration and management of XRDS
3167 synonym elements.

Comment [DSR27]: This section is new in ED06.

3168 17.6 Redirect and Ref Management

3169 As discussed in sections 5.3 and 11, XRI and XRDS infrastructure includes the capability to
3170 distribute and delegate XRDS document management across multiple network locations or
3171 identifier authorities. Identifier authorities should follow the security precautions highlighted in
3172 section 5.3 Redirects and Refs are properly authorized and represent the intended delegation
3173 policies.

Comment [DSR28]: This section is new in ED06.

3174 17.7 Community Root Authorities

3175 The XRI authority information for a community root needs to be well-known to the clients that
3176 request resolution within that community. For trusted resolution, this includes the authority
3177 resolution service endpoint URIs, the `xrd:XRD/xrd:ProviderID`, and the `ds:KeyInfo`
3178 information. An acceptable means of providing this information is for the community root authority
3179 to produce a self-signed XRD and publish it to a server-authenticated HTTPS endpoint. Special
3180 care should be taken to ensure the correctness of such an XRD; if this information is incorrect, an
3181 attacker may be able to convince a client of an incorrect result during trusted resolution.

3182 17.8 Caching Authorities

3183 In addition to traditional HTTP caching proxies, XRI proxy resolvers may be a part of the
3184 resolution topology. Such proxy resolvers should take special precautions against cache
3185 poisoning, as these caching entities may represent trusted decision points within a deployment's
3186 resolution architecture.

3187 17.9 Recursing and Proxy Resolution

3188 During recursing resolution, subsegments of the XRI authority segment for which the resolving
3189 network endpoint is not authoritative may be revealed to that service endpoint. During proxy
3190 resolution, some or all of an XRI is provided to the proxy resolver.

3191 In both cases, privacy considerations should be evaluated before disclosing such information.

3192 17.10 Denial-Of-Service Attacks

3193 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks
3194 typical of systems relying on DNS and HTTP.

A. Acknowledgments

3196 The editors would like to acknowledge the contributions of the OASIS XRI Technical Committee,
 3197 whose voting members at the time of publication were:

- 3198 • Geoffrey Strongin, Advanced Micro Devices
- 3199 • Ajay Madhok, AmSoft Systems
- 3200 • Jean-Jacques Dubray, Attachmate
- 3201 • William Barnhill, Booz Allen and Hamilton
- 3202 • Drummond Reed, Cordance Corporation
- 3203 • Marc Le Maitre, Cordance Corporation
- 3204 • Dave McAlpin, Epok
- 3205 • Loren West, Epok
- 3206 • Peter Davis, NeuStar
- 3207 • Masaki Nishitani, Nomura Research
- 3208 • Nat Sakimura, Nomura Research
- 3209 • Tetsu Watanabe, Nomura Research
- 3210 • Owen Davis, PlaNetwork
- 3211 • Victor Grey, PlaNetwork
- 3212 • Fen Labalme, PlaNetwork
- 3213 • Mike Lindelsee, Visa International
- 3214 • Gabriel Wachob, Visa International
- 3215 • Dave Wentker, Visa International
- 3216 • Bill Washburn, XDI.ORG

Comment [DSR29]: TODO - Needs updating.

3217 The editors also would like to acknowledge the following people for their contributions to previous
 3218 versions of the OASIS XRI specifications (affiliations listed for OASIS members):

- 3219 Thomas Bikeev, EAN International; Krishna Sankar, Cisco; Winston Bumpus, Dell; Joseph
 3220 Moeller, EDS; Steve Green, Epok; Lance Hood, Epok; Adarbad Master, Epok; Davis McPherson,
 3221 Epok; Chetan Sabnis, Epok; Phillipe LeBlanc, GemPlus; Jim Schreckengast, Gemplus; Xavier
 3222 Serret, Gemplus; John McGarvey, IBM; Reva Modi, Infosys; Krishnan Rajagopalan, Novell;
 3223 Tomonori Seki, NRI; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,
 3224 Verisign; Rajeev Maria, Visa International; Terence Spielman, Visa International; John Veizades,
 3225 Visa International; Lark Allen, Wave Systems; Michael Willett, Wave Systems; Matthew Dovey;
 3226 Eamonn Neylon; Mary Nishikawa; Lars Marius Garshol; Norman Paskin; Bernard Vatant.
- 3227 • A special acknowledgement to Jerry Kindall (Epok) for a full editorial review.

B. Non-Normative Text

3229

C. Revision History

Comment [DSR30]: TODO – We need to determine if we need a revision history, and if so, what should be in it.

3230

[optional; should not be included in OASIS Standards]

3231

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>

3232

3233

3234

D. XML Schema for XRDS and XRD (Normative)

```

3235 <?xml version="1.0" encoding="UTF-8"?>
3236 <xs:schema targetNamespace="xri://$xrds" elementFormDefault="qualified"
3237 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds">
3238   <!-- Utility patterns -->
3239   <xs:attributeGroup name="otherattribute">
3240     <xs:anyAttribute namespace="##other" processContents="lax"/>
3241   </xs:attributeGroup>
3242   <xs:group name="otherelement">
3243     <xs:choice>
3244       <xs:any namespace="##other" processContents="lax"/>
3245       <xs:any namespace="##local" processContents="lax"/>
3246     </xs:choice>
3247   </xs:group>
3248   <!-- Patterns for elements -->
3249   <xs:element name="XRDS">
3250     <xs:complexType>
3251       <xs:sequence>
3252         <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3253       </xs:sequence>
3254       <xs:attributeGroup ref="xrds:otherattribute"/>
3255       <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
3256     </xs:complexType>
3257   </xs:element>
3258 </xs:schema>
3259
3260
3261 <?xml version="1.0" encoding="UTF-8"?>
3262 <xs:schema targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified"
3263 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
3264 xmlns:xrd="xri://$xrd*($v*2.0)">
3265   <!-- Utility patterns -->
3266   <xs:attributeGroup name="otherattribute">
3267     <xs:anyAttribute namespace="##other" processContents="lax"/>
3268   </xs:attributeGroup>
3269   <xs:group name="otherelement">
3270     <xs:choice>
3271       <xs:any namespace="##other" processContents="lax"/>
3272       <xs:any namespace="##local" processContents="lax"/>
3273     </xs:choice>
3274   </xs:group>
3275   <xs:attributeGroup name="priorityAttrGrp">
3276     <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
3277   </xs:attributeGroup>
3278   <xs:attributeGroup name="selectionAttrGrp">
3279     <xs:attribute name="match" use="optional" default="default">
3280       <xs:simpleType>
3281         <xs:restriction base="xs:string">
3282           <xs:enumeration value="default"/>
3283           <xs:enumeration value="content"/>
3284           <xs:enumeration value="any"/>
3285           <xs:enumeration value="non-null"/>
3286           <xs:enumeration value="null"/>
3287           <xs:enumeration value="none"/>
3288         </xs:restriction>
3289       </xs:simpleType>
3290     </xs:attribute>
3291     <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
3292   </xs:attributeGroup>
3293   <xs:complexType name="URIPattern">
3294     <xs:simpleContent>
3295       <xs:extension base="xs:anyURI">
3296         <xs:attributeGroup ref="xrd:otherattribute"/>
3297       </xs:extension>
3298     </xs:simpleContent>
3299   </xs:complexType>
3300   <xs:complexType name="URIPriorityPattern">

```

```

3301     <xs:simpleContent>
3302         <xs:extension base="xrd:URIPattern">
3303             <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3304         </xs:extension>
3305     </xs:simpleContent>
3306 </xs:complexType>
3307 <xs:complexType name="StringPattern">
3308     <xs:simpleContent>
3309         <xs:extension base="xs:string">
3310             <xs:attributeGroup ref="xrd:otherattribute"/>
3311         </xs:extension>
3312     </xs:simpleContent>
3313 </xs:complexType>
3314 <xs:complexType name="StringSelectionPattern">
3315     <xs:simpleContent>
3316         <xs:extension base="xrd:StringPattern">
3317             <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
3318         </xs:extension>
3319     </xs:simpleContent>
3320 </xs:complexType>
3321 <!-- Patterns for elements -->
3322 <xs:element name="XRD">
3323     <xs:complexType>
3324         <xs:sequence>
3325             <xs:element ref="xrd:Query" minOccurs="0"/>
3326             <xs:element ref="xrd:Status" minOccurs="0"/>
3327             <xs:element ref="xrd:Expires" minOccurs="0"/>
3328             <xs:element ref="xrd:ProviderID" minOccurs="0"/>
3329             <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
3330             <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded"/>
3331             <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
3332             <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded"/>
3333             <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3334         </xs:sequence>
3335         <xs:attribute name="id" type="xs:ID"/>
3336         <xs:attribute name="idref" type="xs:IDREF" use="optional"/>
3337         <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0"/>
3338         <xs:attributeGroup ref="xrd:otherattribute"/>
3339     </xs:complexType>
3340 </xs:element>
3341 <xs:element name="Query" type="xrd:StringPattern"/>
3342 <xs:element name="Status">
3343     <xs:complexType>
3344         <xs:simpleContent>
3345             <xs:extension base="xrd:StringPattern">
3346                 <xs:attribute name="code" type="xs:int" use="required"/>
3347                 <xs:attributeGroup ref="xrd:otherattribute"/>
3348             </xs:extension>
3349         </xs:simpleContent>
3350     </xs:complexType>
3351 </xs:element>
3352 <xs:element name="Expires">
3353     <xs:complexType>
3354         <xs:simpleContent>
3355             <xs:extension base="xs:dateTime">
3356                 <xs:attributeGroup ref="xrd:otherattribute"/>
3357             </xs:extension>
3358         </xs:simpleContent>
3359     </xs:complexType>
3360 </xs:element>
3361 <xs:element name="ProviderID" type="xrd:URIPattern"/>
3362 <xs:element name="LocalID">
3363     <xs:complexType>
3364         <xs:simpleContent>
3365             <xs:extension base="xrd:StringPattern">
3366                 <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3367             </xs:extension>
3368         </xs:simpleContent>
3369     </xs:complexType>
3370 </xs:element>
3371 <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>

```

```

3372 <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
3373 <xs:element name="Service">
3374   <xs:complexType>
3375     <xs:sequence>
3376       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
3377       <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
3378       <xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
3379       <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
3380       <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
3381       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3382     </xs:sequence>
3383     <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3384     <xs:attributeGroup ref="xrd:otherattribute"/>
3385   </xs:complexType>
3386 </xs:element>
3387 <xs:element name="Type">
3388   <xs:complexType>
3389     <xs:simpleContent>
3390       <xs:extension base="xrd:URIPattern">
3391         <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
3392       </xs:extension>
3393     </xs:simpleContent>
3394   </xs:complexType>
3395 </xs:element>
3396 <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
3397 <xs:element name="Path" type="xrd:StringSelectionPattern"/>
3398 <xs:element name="URI">
3399   <xs:complexType>
3400     <xs:simpleContent>
3401       <xs:extension base="xrd:URIPattern">
3402         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3403         <xs:attribute name="append">
3404           <xs:simpleType>
3405             <xs:restriction base="xs:string">
3406               <xs:enumeration value="none"/>
3407               <xs:enumeration value="local"/>
3408               <xs:enumeration value="authority"/>
3409               <xs:enumeration value="path"/>
3410               <xs:enumeration value="query"/>
3411               <xs:enumeration value="qxri"/>
3412             </xs:restriction>
3413           </xs:simpleType>
3414         </xs:attribute>
3415       </xs:extension>
3416     </xs:simpleContent>
3417   </xs:complexType>
3418 </xs:element>
3419 </xs:schema>
3420
3421

```

3422

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

3423

3424 [TODO]

3425 **F. Media Type Definition for application/xrds+xml**
3426 **(Normative)**

3427 This section is prepared in anticipation of media type registration meeting the requirements of
3428 **[RFC4288]**.

3429 **Type name:** application

3430 **Subtype name:** xrds+xml

3431 **Required parameters:** None

3432 **Optional parameters:** See Table 6 of this document.

3433 **Encoding considerations:** Identical to those of "application/xml" as described in **[RFC3023]**,
3434 Section 3.2.

3435 **Security considerations:** As defined in this specification. In addition, as this media type uses the
3436 "+xml" convention, it shares the same security considerations as described in **[RFC3023]**,
3437 Section 10.

3438 **Interoperability considerations:** There are no known interoperability issues.

3439 **Published specification:** This specification.

3440 **Applications that use this media type:** Applications conforming to this specification use this
3441 media type.

3442 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
3443 Technical Committee Co-Chair, drummond.reed@cordance.net

3444 **Intended usage:** COMMON

3445 **Restrictions on usage:** None

3446 **Author:** OASIS XRI TC

3447 **Change controller:** OASIS XRI TC

3448

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

3449

3450 This section is prepared in anticipation of media type registration meeting the requirements of
3451 [RFC4288].

3452 **Type name:** application

3453 **Subtype name:** xrd+xml

3454 **Required parameters:** None

3455 **Optional parameters:** See Table 6 of this document.

3456 **Encoding considerations:** Identical to those of "application/xml" as described in [RFC3023],
3457 Section 3.2.

3458 **Security considerations:** As defined in this specification. In addition, as this media type uses the
3459 "+xml" convention, it shares the same security considerations as described in [RFC3023],
3460 Section 10.

3461 **Interoperability considerations:** There are no known interoperability issues.

3462 **Published specification:** This specification.

3463 **Applications that use this media type:** Applications conforming to this specification use this
3464 media type.

3465 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
3466 Technical Committee Co-Chair, drummond.reed@cordance.net

3467 **Intended usage:** COMMON

3468 **Restrictions on usage:** None

3469 **Author:** OASIS XRI TC

3470 **Change controller:** OASIS XRI TC

3471

H. Example Local Resolver Interface Definition (Informative)

Comment [DSR32]: Needs updating (issue #38 and new parameter types)

3472

3473 Following is a language-neutral example of an interface definition for a local XRI resolver
3474 consistent with the requirements of this specification.

3475 The interface definition is provided as five operations where each operation takes three or more
3476 of the following input parameters. The input parameters are described here in terms of the
3477 normative text in section 5. In all of these parameters, the value empty string ("") is interpreted the
3478 same as the value null.

3479

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 [DSR33]: Needs updating

3480

3481 The five operations correspond to the following combinations of values of the Resolution Output
3482 Format parameter and its `sep` (service endpoint) subparameter (section 7.1.2) as shown below.

3483

	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

3484 Following is the API and descriptions of the five operations.

3485 1. Resolve Authority to XRDS

```
3486 int resolveAuthToXRDS(  
3487     in string QXRI, in string trustType, in boolean followRefs,  
3488     out string XRDS, out string errorContext);
```

- 3489 • Performs authority resolution only (sections 5 and 6) and outputs the XRDS as specified in
3490 section 4.2.1 when the `sep` subparameter is `false`.
- 3491 • Only the authority segment of the QXRI is processed by this function. If the QXRI contains a
3492 path or query component, it is ignored.
- 3493 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 3494 • Returns the error code. If error, then the `errorContext` output argument may contain additional
3495 error information. The output XRDS will contain a final XRD with the same status code and
3496 optional context information in its `xrd:Status` element.

3497

3498 2. Resolve Authority to XRD

```
3499 int resolveAuthToXRD(  
3500     in string QXRI, in string trustType, in boolean followRefs,  
3501     out string XRD, out string errorContext);
```

- 3502 • Performs authority resolution only (sections 5 and 6) and outputs the final XRD as specified
3503 in section 4.2.2 when the `sep` subparameter is `false`.
- 3504 • Only the authority segment of the QXRI is processed by this function. If the QXRI contains a
3505 path or query component, it is ignored.
- 3506 • The output XRD argument will be signed or not depending on the value of `trustType`.
- 3507 • Returns the error code. If error, then the `errorContext` output argument may contain
3508 additional error information. The output XRD will contain the same status code and optional
3509 context information in its `xrd:Status` element.

3510

3511 3. Resolve Service Endpoint to XRDS

```
3512 int resolveSEPToXRDS(  
3513     in string QXRI, in string trustType, in string sepType,  
3514     in string sepMediaType, in boolean followRefs,  
3515     out string XRDS, out string errorContext);
```

- 3516 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
3517 and outputs the XRDS as specified in section 4.2.1 when the `sep` subparameter is `true`.
- 3518 • As specified in section 4.2.1, the output XRDS document will contain a nested XRDS
3519 document as the final child element if reference processing was necessary to locate the
3520 request service endpoint and the `followRefs` flag was set to `true`.
- 3521 • The final XRD in the output XRD will either contain at least one instance of the requested
3522 service endpoint or an error. IMPORTANT: Although the resolver will perform this verification,
3523 the final XRD is NOT filtered when the Resolution Output Format is an XRDS document.
3524 Filtering is only performed when the Resolution Output Format is an XRD document (see
3525 next section).
- 3526 • The output XRDS argument will be signed or not depending on the value of `trustType`.
- 3527 • Returns the error code. If error, then the `errorContext` output argument may contain
3528 additional error information. The output XRDS will contain a final XRD with the same status
3529 code and optional context information in its `xrd:Status` element.
- 3530 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
3531 same as the value `null`.

3532

3533 4. Resolve Service Endpoint to XRD

```
3534 int resolveSEPToXRD(  
3535     in string QXRI, in string trustType, in string sepType,  
3536     in string sepMediaType, in boolean followRefs,  
3537     out string XRDS, out string errorContext);
```

- 3538 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
3539 and outputs the XRD as specified in section 4.2.2 when the `sep` subparameter is `true`.
- 3540 • As specified in section 4.2.2, all elements in the output XRD subject to the global
3541 `xrd:priority` attribute will be returned in order of highest to lowest.
- 3542 • The output XRD will either contain at least one instance of the requested service endpoint or
3543 an error.
- 3544 • The output XRD will be *not* be signed regardless of the value of `trustType` because the
3545 XRD will be filtered to only the selected service endpoints.
- 3546 • Returns the error code. If error, then the `errorContext` output argument may contain
3547 additional error information. The output XRD will contain the same status code and optional
3548 context information in its `xrd:Status` element.
- 3549 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
3550 same as the value `null`.

3551 **5. Resolve Service Endpoint to URI List**

```
3552 int resolveSepToURIList(  
3553     in string QXRI, in string trustType, in string sepType,  
3554     in string sepMediaType, in boolean followRefs,  
3555     out string[] URIList, out string errorContext);
```

- 3556 • Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8)
3557 and, upon success, outputs a non-empty URI List as specified in section 4.2.3.
- 3558 • Returns the error code. If error, then the output URI List will be empty, and the
3559 `errorContext` output argument may contain additional error information.
- 3560 • For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the
3561 same as the value null.

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 26.
111	CEID_VERIFIED	Any	Both CanonicalID and CanonicalEquiID verification succeeded. See Table 26.
112	CID_NOT_PRESENT	Any	A CanonicalID element was not present. See Table 26.
113	CID_VERIFICATION_FAILED	Any	CanonicalID verification failed. See Table 26.
114	CEID_VERIFICATION_FAILED	Any	CanonicalID verification succeeded but CanonicalEquiID verification failed. See Table 26.