



Extensible Resource Identifier (XRI) Resolution Version 2.0

Committee Draft 02 Revision 03

27 February 2008

Specification URIs:

This Version:

<http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02-rv-03.html>
<http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02-rv-03.pdf>
<http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02-rv-03.doc>

Previous Version:

<http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02.html>
<http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02.pdf>
<http://docs.oasis-open.org/xri/2.0/specs/cd02/xri-resolution-V2.0-cd-02.doc>

Latest Version:

<http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html>
<http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.pdf>
<http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.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>

Related Work:

This specification replaces or supercedes:

- Extensible Resource Identifier (XRI) Resolution Version 2.0, Committee Draft 01, March 2005
- Extensible Resource Identifier (XRI) Version 1.0, Committee Draft 01, January 2004

This specification is related to:

- Extensible Resource Identifier (XRI) Syntax Version 2.0, Committee Specification, December 2005
- Extensible Resource Identifier (XRI) Metadata Version 2.0, Committee Draft 01, March 2005

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 simple 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 (XRI) using XRDS documents and HTTP(S) URIs. These protocols are intended for use with both HTTP(S) URIs as defined in [RFC2616] and with XRI as defined by *Extensible Resource Identifier (XRI) Syntax Version 2.0* [XRISyntax] or higher. For a dictionary of XRI defined to provide standardized identifier metadata, see *Extensible Resource Identifier (XRI) Metadata Version 2.0* [XRIMetadata]. For a basic introduction to XRI, see the *XRI 2.0 FAQ* [XRIFAQ].

Status:

This document was last revised or approved by the XRI Technical Committee on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

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

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

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

Notices

Copyright © OASIS® 1993–2008. All Rights Reserved.

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", "Extensible Resource Identifier", and "XRI" 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.

Table of Contents

1	Introduction.....	11
1.1	Overview of XRI Resolution Architecture.....	11
1.2	Structure of this Specification.....	14
1.3	Terminology and Notation.....	15
1.4	Examples.....	15
1.5	Normative References.....	15
1.6	Non-Normative References.....	16
2	Conformance.....	17
2.1	Conformance Targets.....	17
2.2	Conformance Claims.....	17
2.3	XRDS Clients.....	17
2.4	XRDS Servers.....	17
2.5	XRI Local Resolvers.....	18
2.5.1	Generic.....	18
2.5.2	HTTPS.....	18
2.5.3	SAML.....	18
2.6	XRI Proxy Resolvers.....	18
2.6.1	Generic.....	18
2.6.2	HTTPS.....	18
2.6.3	SAML.....	18
2.7	XRI Authority Servers.....	19
2.7.1	Generic.....	19
2.7.2	HTTPS.....	19
2.7.3	SAML.....	19
2.8	Extensions.....	19
2.9	Language.....	19
3	Namespaces.....	20
3.1	XRI Namespaces for XRI Resolution.....	20
3.1.1	XRIs Reserved for XRI Resolution.....	20
3.1.2	XRIs Assigned to XRI Resolution Service Types.....	20
3.2	XML Namespaces for XRI Resolution.....	21
3.3	Media Types for XRI Resolution.....	21
4	XRDS Documents.....	23
4.1	XRDS and XRD Namespaces and Schema Locations.....	23
4.2	XRD Elements and Attributes.....	23
4.2.1	Management Elements.....	25
4.2.2	Trust Elements.....	26
4.2.3	Synonym Elements.....	27
4.2.4	Service Endpoint Descriptor Elements.....	27
4.2.5	Service Endpoint Trust Elements.....	29
4.2.6	Service Endpoint Selection Elements.....	29
4.3	XRD Attribute Processing Rules.....	30

4.3.1 ID Attribute	30
4.3.2 Version Attribute	30
4.3.3 Priority Attribute	30
4.4 XRI and IRI Encoding Requirements.....	31
5 XRD Synonym Elements	32
5.1 Query Identifiers	32
5.1.1 HTTP(S) URI Query Identifiers.....	32
5.1.2 XRI Query Identifiers.....	32
5.2 Synonym Elements	33
5.2.1 LocalID.....	33
5.2.2 EquivID	33
5.2.3 CanonicalID	34
5.2.4 CanonicalEquivID	34
5.3 Redirect and Ref Elements	35
5.4 XRD Equivalence	35
5.5 Synonym Verification.....	36
5.6 Synonym Selection	36
6 Discovering an XRDS Document from an HTTP(S) URI.....	37
6.1 Overview	37
6.2 HEAD Protocol.....	37
6.3 GET Protocol.....	37
7 XRI Resolution Flow	39
8 Inputs and Outputs.....	41
8.1 Inputs	41
8.1.1 QXRI (Authority String, Path String, and Query String)	43
8.1.2 Resolution Output Format	43
8.1.3 Service Type.....	44
8.1.4 Service Media Type	45
8.2 Outputs	45
8.2.1 XRDS Document.....	47
8.2.2 XRD Element	47
8.2.3 URI List.....	48
8.2.4 HTTP(S) Redirect	48
9 Generic Authority Resolution Service.....	49
9.1 XRI Authority Resolution	49
9.1.1 Service Type and Service Media Type.....	49
9.1.2 Protocol.....	50
9.1.3 Requesting an XRDS Document using HTTP(S)	52
9.1.4 Failover Handling.....	53
9.1.5 Community Root Authorities	54
9.1.6 Self-Describing XRDS Documents.....	55
9.1.7 Qualified Subsegments	55
9.1.8 Cross-References	56
9.1.9 Selection of the Next Authority Resolution Service Endpoint	56
9.1.10 Construction of the Next Authority URI.....	57

9.1.11	Recurring Authority Resolution	57
9.2	IRI Authority Resolution.....	58
9.2.1	Service Type and Media Type	58
9.2.2	Protocol.....	58
9.2.3	Optional Use of HTTPS.....	58
10	Trusted Authority Resolution Service	60
10.1	HTTPS	60
10.1.1	Service Type and Service Media Type	60
10.1.2	Protocol.....	60
10.2	SAML	60
10.2.1	Service Type and Service Media Type	61
10.2.2	Protocol.....	61
10.2.3	Recurring Authority Resolution	62
10.2.4	Client Validation of XRDS.....	63
10.2.5	Correlation of ProviderID and KeyInfo Elements	64
10.3	HTTPS+SAML	64
10.3.1	Service Type and Service Media Type.....	64
10.3.2	Protocol.....	65
11	Proxy Resolution Service	66
11.1	Service Type and Media Types	66
11.2	HXRIs.....	66
11.3	HXRI Query Parameters	67
11.4	HXRI Encoding/Decoding Rules.....	68
11.5	HTTP(S) Accept Headers.....	70
11.6	Null Resolution Output Format	70
11.7	Outputs and HTTP(S) Redirects.....	70
11.8	Differences Between Proxy Resolution Servers	71
11.9	Combining Authority and Proxy Resolution Servers	71
12	Redirect and Ref Processing	72
12.1	Cardinality	74
12.2	Precedence	74
12.3	Redirect Processing	75
12.4	Ref Processing.....	76
12.5	Nested XRDS Documents.....	77
12.5.1	Redirect Examples.....	77
12.5.2	Ref Examples.....	81
12.6	Recursion and Backtracking.....	84
13	Service Endpoint Selection	85
13.1	Processing Rules	85
13.2	Service Endpoint Selection Logic	87
13.3	Selection Element Matching Rules.....	88
13.3.1	Selection Element Match Options	88
13.3.2	The Match Attribute.....	88
13.3.3	Absent Selection Element Matching Rule	89
13.3.4	Empty Selection Element Matching Rule	89

13.3.5 Multiple Selection Element Matching Rule	89
13.3.6 Type Element Matching Rules	89
13.3.7 Path Element Matching Rules	90
13.3.8 MediaType Element Matching Rules	92
13.4 Service Endpoint Matching Rules	92
13.4.1 Service Endpoint Match Options	92
13.4.2 Select Attribute Match Rule	92
13.4.3 All Positive Match Rule	92
13.4.4 Default Match Rule	92
13.5 Service Endpoint Selection Rules	93
13.5.1 Positive Match Rule	93
13.5.2 Default Match Rule	93
13.6 Pseudocode	93
13.7 Construction of Service Endpoint URIs	95
13.7.1 The append Attribute	95
13.7.2 The uric Parameter	96
14 Synonym Verification	97
14.1 Redirect Verification	97
14.2 EquivID Verification	97
14.3 CanonicalID Verification	98
14.3.1 HTTP(S) URI Verification Rules	99
14.3.2 XRI Verification Rules	99
14.3.3 CanonicalEquivID Verification	99
14.3.4 Verification Status Attributes	100
14.3.5 Examples	101
15 Status Codes and Error Processing	106
15.1 Status Elements	106
15.2 Status Codes	106
15.3 Status Context Strings	109
15.4 Returning Errors in Plain Text or HTML	109
15.5 Error Handling in Recursing and Proxy Resolution	109
16 Use of HTTP(S)	110
16.1 HTTP Errors	110
16.2 HTTP Headers	110
16.2.1 Caching	110
16.2.2 Location	110
16.2.3 Content-Type	110
16.3 Other HTTP Features	110
16.4 Caching and Efficiency	111
16.4.1 Resolver Caching	111
16.4.2 Synonyms	111
17 Extensibility and Versioning	112
17.1 Extensibility	112
17.1.1 Extensibility of XRDS	112
17.1.2 Other Points of Extensibility	114

17.2	Versioning	114
17.2.1	Version Numbering	114
17.2.2	Versioning of the XRI Resolution Specification	114
17.2.3	Versioning of Protocols	115
17.2.4	Versioning of XRDs.....	115
18	Security and Data Protection	116
18.1	DNS Spoofing or Poisoning.....	116
18.2	HTTP Security.....	116
18.3	SAML Considerations.....	116
18.4	Limitations of Trusted Resolution	116
18.5	Synonym Verification.....	117
18.6	Redirect and Ref Management.....	117
18.7	Community Root Authorities.....	117
18.8	Caching Authorities	117
18.9	Recurring and Proxy Resolution	117
18.10	Denial-Of-Service Attacks	117
A.	Acknowledgments.....	118
B.	RelaxNG Schema for XRDS and XRD.....	119
C.	XML Schema for XRDS and XRD.....	122
D.	Media Type Definition for application/xrds+xml.....	126
E.	Media Type Definition for application/xrd+xml.....	127
F.	Example Local Resolver Interface Definition.....	128
G.	Revision History.....	132

Table of Figures

Figure 1: Four typical scenarios for XRI authority resolution.....	12
Figure 2: Top-level flowchart of XRI resolution phases.....	39
Figure 3: Input processing flowchart.....	42
Figure 4: Output processing flowchart.....	46
Figure 5: Authority resolution flowchart.....	50
Figure 6: XRDS request flowchart.....	52
Figure 7: Redirect and Ref processing flowchart.....	73
Figure 8: Service endpoint (SEP) selection flowchart.....	85
Figure 9: Service endpoint (SEP) selection logic flowchart.....	87

Table of Tables

Table 1: Comparing DNS and XRI resolution architecture.....	11
Table 2: XRIs reserved for XRI resolution.	20
Table 3: XRIs assigned to identify XRI resolution service types.	20
Table 4: XML namespace prefixes used in this specification.....	21
Table 5: Media types defined or used in this specification.	21
Table 6: Parameters for the media types defined in Table 5.....	22
Table 7: The four XRD synonym elements.	32
Table 8: Input parameters for XRI resolution.	41
Table 9: Subparameters of the QXRI input parameter.....	43
Table 10: Outputs of XRI resolution.....	45
Table 11: Service Type and Service Media Type values for generic authority resolution.	49
Table 12: Parsing the first subsegment of an XRI that begins with a global context symbol.....	56
Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.	56
Table 14: Examples of the Next Authority URIs constructed using different types of cross-references. ...	56
Table 15: Service Type and Service Media Type values for HTTPS trusted authority resolution.....	60
Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.....	61
Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution..	64
Table 18: Service Type and Service Media Type values for proxy resolution.	66
Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.....	67
Table 20: Example of HXRI components prior to transformation to URI-normal form.	69
Table 21: Example of HXRI components after transformation to URI-normal form.....	69
Table 22: Example of HXRI components after application of the required encoding rules.....	69
Table 23: Comparison of Redirect and Ref elements.	72
Table 24: Match options for selection elements.....	88
Table 25: Enumerated values of the global match attribute and corresponding matching rules.....	88
Table 26: Examples of applying the Path element matching rules.....	91
Table 27: Match options for service endpoints.	92
Table 28: Values of the <code>append</code> attribute and the corresponding QXRI component to append.....	95
Table 29: Error codes for XRI resolution.....	108

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, database keys, filenames, 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 requesting XRDS documents using HTTP(S) URIs, and standard protocol for resolving XRIs using XRDS documents and HTTP(S) URIs. 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.

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 a set of resource records describing a host. If the resolver does not have the answer cached, it will start by querying one of the well-known DNS root nameservers for the fully qualified domain name. Since domain names work from right to left, and the root nameservers know only about top level domains, they will return the NS (name server) records for the top-level domain. The resolver will then repeat the same query to those name servers and “walk down the tree” until the domain name is fully resolved or an error is encountered.

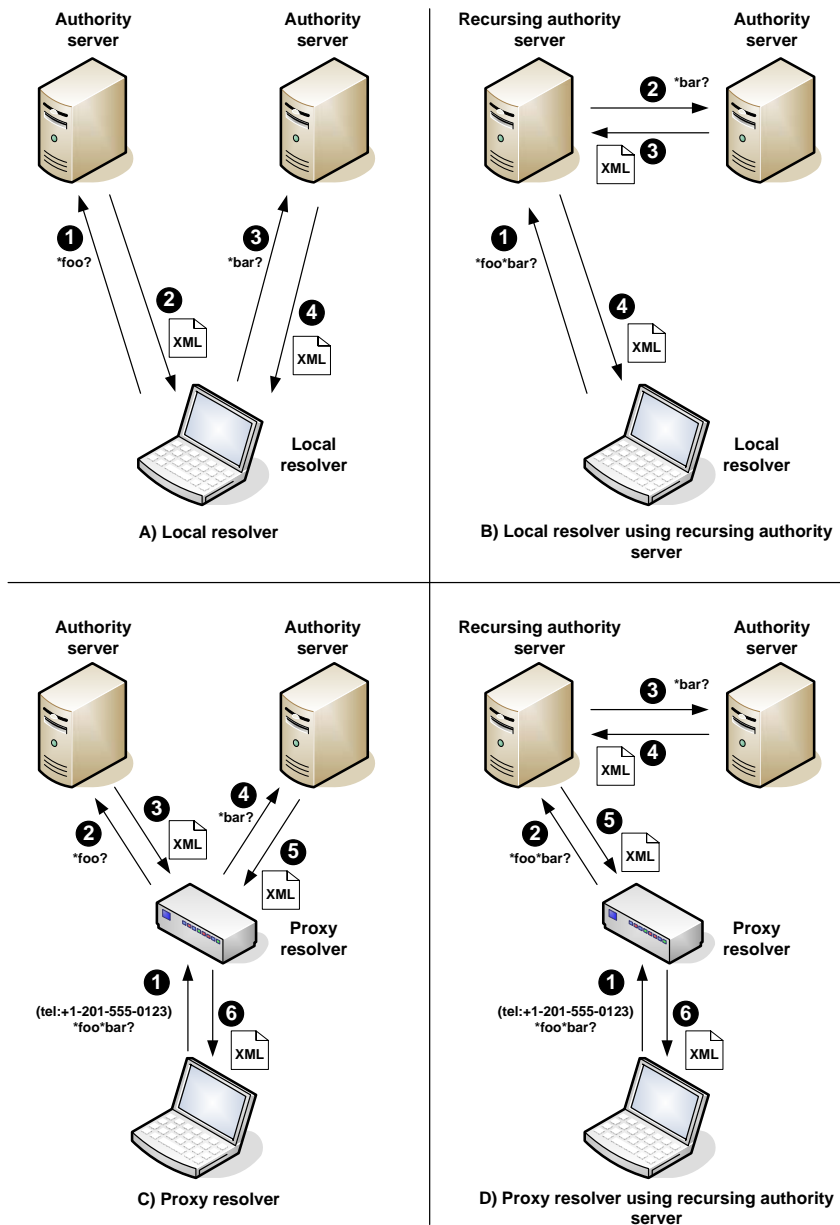
A simple *non-recursing resolver* will rely on a *recursing nameserver* to do this work. For example, it will send a query for the fully qualified domain name `docs.oasis-open.org` to a local nameserver. If the nameserver doesn't have the answer cached, it will resolve the domain name and return the results back to the resolver (and cache the results for subsequent queries).

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 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 server
Recursing resolution	recursing nameserver	recursing authority server or proxy resolver

Table 1: Comparing DNS and XRI resolution architecture.

31 As Table 1 notes, XRI resolution architecture supports both recursing authority servers and *proxy*
 32 *resolvers*. A proxy resolver is simply an HTTP(S) interface to a local XRI resolver (one
 33 implemented using a platform-specific API). Proxy resolvers enable applications—even those that
 34 only understand HTTP URIs—to easily access the functions of an XRI resolver remotely.
 35 Figure 1 shows four scenarios of how these components might interact to resolve
 36 `xri://(tel:+1-201-555-0123)*foo*bar` (unlike DNS, this works from left-to-right).



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

39 Each of these scenarios may involve two phases of XRI resolution:

- 40 • *Phase 1: Authority resolution.* This is the phase required to resolve the authority component
41 of an XRI into an XRDS document describing the target authority. Authority resolution works
42 iteratively from left-to-right across each subsegment in the authority component of the XRI. In
43 XRIs, subsegments are delimited using either a specified set of symbol characters or
44 parentheses. For example, in the XRI `xri://(tel:+1-201-555-0123)*foo*bar`, the
45 authority subsegments are `(tel:+1-201-555-0123)` (the community root authority, in this
46 case a URI expressed as an cross-reference delimited with parentheses), `*foo`, (the first
47 resolvable subsegment), and `*bar`, (the second resolvable subsegment). Note that a
48 resolver must be preconfigured (or have its own way of discovering) the community root
49 authority starting point, so the community root subsegment is not resolved except in one
50 special case (see section 9.1.6).
- 51 • *Phase 2: Optional service endpoint selection.* Once authority resolution is complete, there is
52 an optional second phase of XRI resolution to select a specific type of metadata from the final
53 XRDS document retrieved called a *service endpoint* (SEP). Service endpoints are descriptors
54 of concrete URIs at which network services are available for the target resource. Additional
55 XRI resolution parameters as well as the path component of an XRI may be used as service
56 endpoint selection criteria.

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

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

84 1.2 Structure of this Specification

85 This specification is structured into the following sections:

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

90 The next three sections cover XRDS documents and the requirements for XRDS clients and
91 servers:

- 92 • *XRDS Documents* (section 4) specifies a simple, flexible XML-based container for XRI
93 resolution metadata, service endpoints, and/or other metadata describing a resource.
- 94 • *XRDS Synonyms* (section 5) specifies usage of the four XRDS synonym elements.
- 95 • *Discovering an XRDS Document from an HTTP(S) URI* (section 6) specifies a protocol for
96 obtaining an XRDS description of a resource by starting from an HTTP(S) URI identifying the
97 resource.

98 The remaining sections cover XRI resolution and the requirements for XRI authority servers, local
99 resolvers, and proxy resolvers:

- 100 • *XRI Resolution Flow* (section 7) provides a top-level flowchart of the XRI resolution function
101 together with a list of other supporting flowcharts used throughout the specification.
- 102 • *Inputs and Outputs* (section 8) specifies the input parameters, output formats, and associated
103 processing rules.
- 104 • *Generic Authority Resolution* (section 9) specifies a simple resolution protocol for the
105 authority component of an XRI using HTTP/HTTPS as a transport.
- 106 • *Trusted Authority Resolution* (section 10) specifies three extensions to generic authority
107 resolution for creating a chain of trust between the participating identifier authorities using
108 HTTPS connections, SAML assertions, or both.
- 109 • *Proxy Resolution* (section 11) specifies an HTTP(S) interface for an XRI resolver plus a
110 format for expressing an XRI as an HTTP(S) URI to provide backwards compatibility with
111 existing HTTP(S) infrastructure.
- 112 • *Redirect and Ref Processing* (section 12) specifies how a resolver follows a reference from
113 one XRDS document to another to enable federation of XRDS documents across multiple
114 network locations (Redirects) or identifier authorities (Refs).
- 115 • *Service Endpoint Selection* (section 13) specifies an optional second phase of resolution for
116 selecting a set of service endpoints from an XRDS document.
- 117 • *Synonym Verification* (section 14) specifies how a resolver can verify that one XRI, IRI, or
118 HTTP(S) URI is an authorized synonym for another.
- 119 • *Status Codes and Error Processing* (section 15) specifies status reporting and error handling.
- 120 • *Use of HTTP(S)* (section 16) specifies how the XRDS and XRI resolution protocols leverage
121 features of the HTTP(S) protocol.
- 122 • *Extensibility and Versioning* (section 17) describes how the XRI resolution protocol can be
123 easily extended and how new versions will be identified and accommodated.
- 124 • *Security and Data Protection* (section 18) summarizes key security and privacy
125 considerations for XRI resolution infrastructure.

126 1.3 Terminology and Notation

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

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

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

135 Formatting conventions used in this document:

136 Examples look like this.

137 ABNF productions look like this.

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

139 1.4 Examples

140 The specification includes short examples as necessary to clarify interpretation. However, to
141 minimize non-normative material, it does not include extensive examples of XRI resolution
142 requests and responses. Many such examples are available via open source implementations,
143 operating XRI registry and resolution services, and public websites about XRI. For a list of such
144 resources, see the Wikipedia page on XRI [WikipediaXRI].

145 1.5 Normative References

- 146 [DNSSEC] D. Eastlake, *Domain Name System Security Extensions*,
147 <http://www.ietf.org/rfc/rfc2535>, IETF RFC 2535, March 1999.
- 148 [RFC2045] N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
149 *Part One: Format of Internet Message Bodies*,
150 <http://www.ietf.org/rfc/rfc2045.txt>, IETF RFC 2045, November 1996.
- 151 [RFC2046] N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*
152 *Part Two: Media Types*, <http://www.ietf.org/rfc/rfc2046.txt>, IETF RFC
153 2046, November 1996.
- 154 [RFC2119] S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*,
155 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 156 [RFC2141] R. Moats, *URN Syntax*, <http://www.ietf.org/rfc/rfc2141.txt>, IETF RFC
157 2141, May 1997.
- 158 [RFC2483] M. Mealling, R. Daniel Jr., *URI Resolution Services Necessary for URN*
159 *Resolution*, <http://www.ietf.org/rfc/rfc2483.txt>, IETF RFC 2483, January
160 1999.
- 161 [RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.
162 Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*,
163 <http://www.ietf.org/rfc/rfc2616.txt>, IETF RFC 2616, June 1999.
- 164 [RFC2818] E. Rescorla, *HTTP over TLS*, <http://www.ietf.org/rfc/rfc2818.txt>, IETF
165 RFC 2818, May 2000.
- 166 [RFC3023] M. Murata, S. St-Laurent, D. Kohn, *XML Media Types*,
167 <http://www.ietf.org/rfc/rfc3023.txt>, IETF RFC 3023, January 2001.
- 168 [RFC3986] T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier*
169 *(URI): Generic Syntax*, <http://www.ietf.org/rfc/rfc3986.txt>, IETF RFC
170 3986, January 2005.

171	[RFC4234]	D. H. Crocker and P. Overell, <i>Augmented BNF for Syntax Specifications: ABNF</i> , http://www.ietf.org/rfc/rfc4234.txt , IETF RFC 4234, October 2005.
172		
173	[RFC4288]	N. Freed, J. Klensin, <i>Media Type Specifications and Registration Procedures</i> , http://www.ietf.org/rfc/rfc4288.txt , IETF RFC 4288, December 2005.
174		
175		
176	[SAML]	S. Cantor, J. Kemp, R. Philpott, E. Maler, <i>Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0</i> , http://www.oasis-open.org/committees/security , March 2005.
177		
178		
179	[Unicode]	The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1 (http://www.unicode.org/versions/Unicode4.0.1) and by Unicode 4.1.0 (http://www.unicode.org/versions/Unicode4.1.0), March, 2005.
180		
181		
182		
183		
184	[UUID]	Open Systems Interconnection – <i>Remote Procedure Call</i> , ISO/IEC 11578:1996, http://www.iso.org/ , August 2001.
185		
186	[XML]	T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, <i>Extensible Markup Language (XML) 1.0, Third Edition</i> , World Wide Web Consortium, http://www.w3.org/TR/REC-xml/ , February 2004.
187		
188		
189	[XMLDSig]	D. Eastlake, J. Reagle, D. Solo et al., <i>XML-Signature Syntax and Processing</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlsig-core/ , February, 2002.
190		
191		
192	[XMLID]	J. Marsh, D. Veillard, N. Walsh, <i>xml:id Version 1.0</i> , World Wide Web Consortium, http://www.w3.org/TR/2005/REC-xml-id-20050909 , September 2005.
193		
194		
195	[XMLSchema]	H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, <i>XML Schema Part 1: Structures Second Edition</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlschema-1/ , October 2004.
196		
197		
198	[XMLSchema2]	P. Biron, A. Malhotra, <i>XML Schema Part 2: Datatypes Second Edition</i> , World Wide Web Consortium, http://www.w3.org/TR/xmlschema-2/ , October 2004.
199		
200		
201	[XRIMetadata]	D. Reed, <i>Extensible Resource Identifier (XRI) Metadata V2.0</i> , http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf , March 2005.
202		
203		
204	[XRISyntax]	D. Reed, D. McAlpin, <i>Extensible Resource Identifier (XRI) Syntax V2.0</i> , http://www.oasis-open.org/committees/download.php/15377 , November 2005.
205		
206		

Comment [DSR1]: This reference was updated to the Committee Specification, but still may not have the right link. TODO – Check with Mary McRae.

207 1.6 Non-Normative References

208	[XRIFAQ]	OASIS XRI Technical Committee, <i>XRI 2.0 FAQ</i> , http://www.oasis-open.org/committees/xri/faq.php , Work-In-Progress, March 2006.
209		
210	[XRIReqs]	G. Wachob, D. Reed, M. Le Maitre, D. McAlpin, D. McPherson, <i>Extensible Resource Identifier (XRI) Requirements and Glossary v1.0</i> , http://www.oasis-open.org/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc , June 2003.
211		
212		
213		
214		
215	[WikipediaXRI]	Wikipedia entry on XRI (Extensible Resource Identifier), http://en.wikipedia.org/wiki/XRI , Wikipedia Foundation.
216		
217	[Yadis]	J. Miller, <i>Yadis Specification Version 1.0</i> , http://yadis.org/ , March 2006.

218 2 Conformance

219 This section specifies the conformance targets of this specification and the requirements that
220 apply to each of them.

221 2.1 Conformance Targets

222 The conformance targets of this specification are:

- 223 1. *XRDS clients*, which provide a limited subset of the functionality of XRI resolvers.
- 224 2. *XRDS servers*, which provide a limited subset of the functionality of XRI authority servers.
- 225 3. *XRI local resolvers*, which may implement any combination of the generic, HTTPS, or
226 SAML resolution protocols.
- 227 4. *XRI proxy resolvers*, which may implement any combination of the generic, HTTPS, or
228 SAML resolution protocols.
- 229 5. *XRI authority servers*, which may implement any combination of the generic, HTTPS, or
230 SAML resolution protocols.

231 Note that a single implementation may serve any combination of these functions. For example, an
232 XRI authority server may also function as an XRDS client and server and an XRI local and proxy
233 resolver.

234 2.2 Conformance Claims

235 A claim of conformance with this specification **MUST** meet the following requirements:

- 236 1. It **MUST** state which conformance targets it implements.
- 237 2. If the conformance target is an XRI local resolver, XRI proxy resolver, or XRI authority
238 server, it **MUST** state which resolution protocols are supported, i.e., generic, HTTPS,
239 and/or SAML.

240 2.3 XRDS Clients

241 An implementation conforms to this specification as an XRDS client if it meets the following
242 conditions:

- 243 1. It **MAY** implement parsing of XRDS Documents as specified in section 4.
- 244 2. It **MUST** implement the client requirements of the XRDS request protocol specified in
245 section 6.

246 2.4 XRDS Servers

247 An implementation conforms to this specification as an XRDS server if it meets the following
248 conditions:

- 249 1. It **MUST** produce valid XRDS Documents as specified in section 4.
- 250 2. It **MUST** implement the server requirements of the XRDS request protocol specified in
251 section 6.

252 **2.5 XRI Local Resolvers**

253 **2.5.1 Generic**

254 An implementation conforms to this specification as a generic local resolver if it meets the
255 following conditions:

- 256 1. It parses XRDS documents as specified in section 4.
- 257 2. It processes resolution inputs and outputs as specified in section 8.
- 258 3. It implements the resolver requirements of the generic resolution protocol specified in
259 section 9.
- 260 4. It implements the Redirect and Ref processing rules specified in section 12.
- 261 5. It implements the Service Endpoint Selection processing rules specified in section 13.
- 262 6. It implements the Synonym Verification processing rules specified in section 14.
- 263 7. It implements the Status Code and Error Processing rules specified in section 15.
- 264 8. It follows the HTTP(S) usage recommendations specified in section 16.

265 **2.5.2 HTTPS**

266 An implementation conforms to this specification as an HTTPS local resolver if it meets all the
267 requirements of a generic local resolver plus the following conditions:

- 268 1. It implements the resolver requirements of the HTTPS trusted resolution protocol
269 specified in section 10.1.

270 **2.5.3 SAML**

271 An implementation conforms to this specification as a SAML local resolver if it meets all the
272 requirements of a generic local resolver plus the following conditions:

- 273 1. It implements the resolver requirements of the SAML trusted resolution protocol specified
274 in section 10.2.
- 275 2. It SHOULD also meet the requirements of an HTTPS local resolver. This is STRONGLY
276 RECOMMENDED for confidentiality of SAML interactions.

277 **2.6 XRI Proxy Resolvers**

278 **2.6.1 Generic**

279 An implementation conforms to this specification as a generic proxy resolver if it meets all the
280 requirements of a generic local resolver plus the following conditions:

- 281 1. It implements the requirements for a proxy resolver specified in section 11.

282 **2.6.2 HTTPS**

283 An implementation conforms to this specification as a HTTPS proxy resolver if it meets all the
284 requirements of a HTTPS local resolver plus the following conditions:

- 285 1. It implements the requirements for a HTTPS proxy resolver specified in section 11.

286 **2.6.3 SAML**

287 An implementation conforms to this specification as a SAML proxy resolver if it meets all the
288 requirements of a SAML local resolver plus the following conditions:

- 289 1. It implements the requirements for a proxy resolver specified in section 11.

- 290 2. It SHOULD also meet the requirements of an HTTPS proxy resolver. This is STRONGLY
291 RECOMMENDED for confidentiality of SAML interactions.

292 **2.7 XRI Authority Servers**

293 **2.7.1 Generic**

294 An implementation conforms to this specification as a generic authority server if it meets the
295 following conditions:

- 296 1. It produces XRDS documents as specified in section 4.
- 297 2. It assigns XRDS synonyms as specified in section 5.
- 298 3. It processes resolution inputs and outputs as specified in section 8.
- 299 4. It implements the server requirements of the generic resolution protocol specified in
300 section 9.
- 301 5. It implements the Status Code and Error Processing rules specified in section 15.
- 302 6. It follows the HTTP(S) usage recommendations specified in section 16.

303 **2.7.2 HTTPS**

304 An implementation conforms to this specification as an HTTPS authority server if it meets all the
305 requirements of a generic authority server plus the following conditions:

- 306 1. It implements the server requirements of the HTTPS trusted resolution protocol specified
307 in section 10.1.

308 **2.7.3 SAML**

309 An implementation conforms to this specification as an SAML authority server if it meets all the
310 requirements of a generic authority server plus the following conditions:

- 311 1. It implements the server requirements of the SAML trusted resolution protocol specified
312 in section 10.2.
- 313 2. It SHOULD also meet the requirements of an HTTPS authority server. This is
314 STRONGLY RECOMMENDED for confidentiality of SAML interactions.

315 **2.8 Extensions**

316 The protocols and XML documents defined in this specification MAY be extended. To maintain
317 interoperability, extensions MUST use the extensibility architecture specified in section 17.

318 Extensions MUST NOT be implemented in a manner that would cause them to be non-
319 interoperable with implementations that do not implement the extensions.

320 **2.9 Language**

321 This specification's normative language is English. Translation into other languages is
322 encouraged.

323 3 Namespaces

324 3.1 XRI Namespaces for XRI Resolution

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

329 3.1.1 XRIs Reserved for XRI Resolution

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

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

332 Table 2: XRIs reserved for XRI resolution.

333 3.1.2 XRIs Assigned to XRI Resolution Service Types

334 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	9
xri://\$res*auth*(\$v*2.0)	Version 2.0 of above	9
xri://\$res*proxy	HTTP(S) proxy resolution service	11
xri://\$res*proxy*(\$v*2.0)	Version 2.0 of above	11

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

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

339 **3.2 XML Namespaces for XRI Resolution**

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

Prefix	XML Namespace	Reference
xs	http://www.w3.org/2001/XMLSchema	[XMLSchema]
saml	urn:oasis:names:tc:SAML:2.0:assertion	[SAML]
ds	http://www.w3.org/2000/09/xmldsig#	[XMLDSig]
xrds	xri://\$xrds	Section 3.1.1 of this document
xrd	xri://\$xrd*(\$v*2.0)	Section 3.1.1 of this document

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

343 **3.3 Media Types for XRI Resolution**

344 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as
 345 defined by [RFC2046], particularly in HTTP Accept headers as specified in [RFC2616]. Table 5
 346 specifies the media types used for XRI resolution. Note that in XRI authority resolution, these
 347 media types MUST be passed as HTTP Accept header values. By contrast, in XRI proxy
 348 resolution these media types MUST be passed as query parameters in an HTTP(S) URI as
 349 specified in section 11.

Media Type	Usage	Reference
application/xrds+xml	Content type for returning the full XRDS document describing a resolution chain	Appendix D
application/xrd+xml	Content type for returning only the final XRD element in a resolution chain	Appendix E
text/uri-list	Content type for returning a list of URIs output from the service endpoint selection process defined in section 12	Section 5 of [RFC2483]

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

351 To provide full control of XRI resolution, the media types specified in Table 5 accept the media
 352 type parameters defined in Table 6. All are Boolean flags. Note that when these media type
 353 parameters are appended to a media type in the XRI proxy resolver interface, the semicolon
 354 character used to concatenate them MUST be percent-encoded as specified in section 11.4.

Media Type Parameter	Default Value	Usage	See Section
https	FALSE	Specifies use of HTTPS trusted resolution	10.1
saml	FALSE	Specifies use of SAML trusted resolution	10.2
refs	TRUE	Specifies whether Refs should be followed during resolution (by default they are followed)	12.4
sep	FALSE	Specifies whether service endpoint selection should be performed	13
nodefault_t	TRUE	Specifies whether a default match on a Type service endpoint selection element is allowed	13.3
nodefault_p	TRUE	Specifies whether a default match on a Path service endpoint selection element is allowed	13.3
nodefault_m	TRUE	Specifies whether a default match on a MediaType service endpoint selection element is allowed	13.3
uric	FALSE	Specifies whether a resolver should automatically construct service endpoint URIs	13.7.1
cid	TRUE	Specifies whether automatic canonical ID verification should be performed	14.3

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

357 When used as logical XRI resolution input parameters, these media type parameters will be
 358 referred to as *subparameters*.

359

4 XRDS Documents

360
361
362
363
364

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) elements. While this specification defines only the XRD elements necessary to support XRI resolution, XRD elements can easily be extended to publish any form of metadata about the resources they describe.

365

4.1 XRDS and XRD Namespaces and Schema Locations

366
367
368
369
370

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 two attributes, `redirect` and `ref`, that are used to identify the resource described by the XRDS document. Both are of type `anyURI`. Use of these attributes is defined in section 12.5.

371
372
373
374
375

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].

376
377

The attributes defined in both the XRDS and XRD schemas are not namespace qualified. In order to prevent conflicts, attributes defined in extensions MUST be namespace qualified.

378
379
380
381

This namespace architecture enables the XRDS namespace to remain constant while allowing the XRD namespace (and the namespaces of other XML elements that may be included in an XRDS document) to be versioned over time. See section 17.2 for more about versioning of the XRD schema.

382
383

The locations of the normative RelaxNG schema files for an XRDS document and an XRD element as defined by this specification are:

384
385

- <http://docs.oasis-open.org/xri/2.0/specs/cd03/xrds.rnc>
- <http://docs.oasis-open.org/xri/2.0/specs/cd03/xrd.rnc>

Comment [DSR2]: New references per the change in 4.2.1.

386

The following URIs will always reference the latest versions of these files:

387
388

- <http://docs.oasis-open.org/xri/2.0/specs/xrds.rnc>
- <http://docs.oasis-open.org/xri/2.0/specs/xrd.rnc>

389
390

A reference listing of each of these files is provided in Appendix B, and a reference listing of the informative W3C XML Schema versions is provided in Appendix C.

391

4.2 XRD Elements and Attributes

392
393
394
395

The following example XRDS instance document illustrates the elements and attributes defined in the XRD schema. Note that because it is provided by the community root authority (`tel:+1-201-555-0123`), it includes only one XRD describing the subsegment `*foo`. Examples in later sections show multiple XRDs.

```

397 <XRDS xmlns="xri://$xrd$" ref="xri://(tel:+1-201-555-0123)*foo">
398   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
399     <Query>*foo</Query>
400     <Status code="100"/>
401     <ServerStatus code="100"/>
402     <Expires>2005-05-30T09:30:10Z</Expires>
403     <ProviderID>xri://(tel:+1-201-555-0123)</ProviderID>
404     <LocalID>*baz</LocalID>
405     <EquivID>https://example.com/example/resource/</EquivID>
406     <CanonicalID>xri://(tel:+1-201-555-0123)!1234</CanonicalID>
407     <CanonicalEquivID>
408       xri://=!4a76!c2f7!9033.78bd
409     </CanonicalEquivID>
410     <Service>
411       <ProviderID>
412         xri://(tel:+1-201-555-0123)!1234
413       </ProviderID>
414       <Type>xri://$res*auth*($v*2.0)</Type>
415       <MediaType>application/xrds+xml</MediaType>
416       <URI priority="10">http://resolve.example.com</URI>
417       <URI priority="15">http://resolve2.example.com</URI>
418       <URI>https://resolve.example.com</URI>
419     </Service>
420     <Service>
421       <ProviderID>
422         xri://(tel:+1-201-555-0123)!1234
423       </ProviderID>
424       <Type>xri://$res*auth*($v*2.0)</Type>
425       <MediaType>application/xrds+xml;https=true</MediaType>
426       <URI>https://resolve.example.com</URI>
427     </Service>
428     <Service>
429       <Type match="null" />
430       <Path select="true">/media/pictures</Path>
431       <MediaType select="true">image/jpeg</MediaType>
432       <URI append="path" >http://pictures.example.com</URI>
433     </Service>
434     <Service>
435       <Type match="null" />
436       <Path select="true">/media/videos</Path>
437       <MediaType select="true">video/mpeg</MediaType>
438       <URI append="path" >http://videos.example.com</URI>
439     </Service>
440     <Service>
441       <ProviderID> xri://!1000!1234.5678</ProviderID>
442       <Type match="null" />
443       <Path match="default" />
444       <URI>http://example.com/local</URI>
445     </Service>
446     <Service>
447       <Type>http://example.com/some/service/v3.1</Type>
448       <URI>http://example.com/some/service/endpoint</URI>
449       <LocalID>https://example.com/example/resource/</LocalID>
450     </Service>
451   </XRD>
452 </XRDS>

```

453 A link to the normative RelaxNG schema definition of the XRD schema is provided in Appendix B.
454 Additional normative requirements that cannot be captured in XML schema notation are specified
455 in the following sections. In the case of any conflict, the normative text in this section shall prevail.

Comment [DSR3]: Throughout this and the following sections, all attribute definitions have been moved to bullet points following the element definition.

4.2.1 Management Elements

The first set of elements are used to manage XRDs, particularly from the perspective of caching, error handling, and delegation. Note that to prevent processing conflicts, the XRD schema permits a choice of either `xrd:XRD/xrd:Redirect` elements or `xrd:XRD/xrd:Ref` elements but not both.

xrd:XRD

Container element for all other XRD elements. Attributes:

- `xml:id` (type `xs:ID`). OPTIONAL except in trusted resolution where it is REQUIRED to uniquely identify this element within the containing `xrds:XRDS` document. See sections 4.3.1 and 12.5. Note that this attribute is not explicitly declared in the normative schema as it is an implicit XML attribute defined in [XMLID].
- `idref` (type `xs:idref`). OPTIONAL except in trusted resolution where it is REQUIRED when an XRD element in a nested `xrd:XRDS` document must reference a previously included XRD instance. See sections 4.3.1 and 12.5.
- `version` (type `xs:string`). OPTIONAL for uses outside of XRI resolution but REQUIRED for XRI resolution as defined in section 4.3.2.

xrd:XRD/xrd:Type

0 or more per `xrd:XRD` element. A unique identifier of type `xs:anyURI` that identifies the type of this XRD. This element is provided to support XRD extensibility as described in section 17.1.1. If no instances of this element are present, the type is defined by this specification. If one or more instances of this element are present, the requirements of the specified XRD type SHOULD be defined by an extension specification, which SHOULD be dereferenceable from the URI, IRI, or XRI used as the value of this element. In all cases XRD processors MAY ignore instances of this element and process the XRD as specified in this document.

Comment [DSR4]: New for XRD extensibility per the comment from Eran Hammer-Lahav.

xrd:XRD/xrd:Query

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

xrd:XRD/xrd:Status

0 or 1 per `xrd:XRD` element. RECOMMENDED for all XRDs. REQUIRED if the resolver must report certain error conditions. The contents of the element are a human-readable message string describing the status of the response as determined by the resolver. For XRI resolution, values of the Status element are defined in section 15. Attributes:

- `code` (type `xs:int`). REQUIRED. Provides a numeric status code. See section 15.
- `cid` (type `xs:enumeration`). OPTIONAL except when REQUIRED to report the results of CanonicalID verification as defined in section 14.3.4.
- `ceid` (type `xs:enumeration`). OPTIONAL except when REQUIRED to report the results of CanonicalID verification as defined in section 14.3.4.

xrd:XRD:xrdServerStatus

0 or 1 per `xrd:XRD` element. Identical to `xrd:XRD/xrd:Status` except this element is used by an XRI authority server to report the status of a resolution request to an XRI resolver, and it does not include the `cid` and `ceid` attributes. See section 15.1. Attributes:

- `code` (type `xs:int`). REQUIRED. Provides a numeric status code. See section 15.

501 **xrd:XRD/xrd:Expires**
502 0 or 1 per `xrd:XRD` element. The date/time, in the form of `xs:dateTime`, after which
503 this XRD cannot be relied upon. To promote interoperability, this date/time value
504 SHOULD use the UTC "Z" time zone and SHOULD NOT use fractional seconds. A
505 resolver MUST NOT use an XRD after the time stated here. A resolver MAY discard this
506 XRD before the time indicated in this result. If the HTTP transport caching semantics
507 specify an expiry time earlier than the time expressed in this attribute, then a resolver
508 MUST NOT use this XRD after the expiry time declared in the HTTP headers per section
509 13.2 of [RFC2616]. See section 16.2.1.

510 **xrd:XRD/xrd:Redirect**
511 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST contain an absolute HTTP(S)
512 URI. Choice between this or the `xrd:XRD/xrd:Ref` element below. MUST be
513 processed by a resolver to locate another XRDS document authorized to describe the
514 target resource as defined in section 12. Attributes:

- 515 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.
- 516 • `append` (type `xs:enumeration`). OPTIONAL. Governs construction of the final
517 redirect URI as defined in section 13.7.

518 **xrd:XRD/xrd:Ref**
519 0 or more more per `xrd:XRD` element. Type `xs:anyURI`. MUST contain an absolute
520 XRI. Choice between this or the `xrd:XRD/xrd:Redirect` element above. MUST be
521 processed by a resolver (depending on the value of the `refs` subparameter) to locate
522 another XRDS document authorized to describe the target resource as defined in section
523 12. Attributes:

- 524 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

525 4.2.2 Trust Elements

526 The second set of elements are for applications where trust must be established in the identifier
527 authority providing the XRD. These elements are OPTIONAL for generic authority resolution
528 (section 9), but may be REQUIRED for specific types of trusted authority resolution (section 10)
529 and CanonicalID verification (section 14.3).

530 **xrd:XRD/xrd:ProviderID**

531 0 or 1 per `xrd:XRD` element. A unique identifier of type `xs:anyURI` for the parent
532 authority providing this XRD. The value of this element MUST be a persistent identifier.
533 There MUST be negligible probability that the value of this element will be assigned as an
534 identifier to any other authority. It is RECOMMENDED to use a fully persistent XRI as
535 defined in [XRISyntax]. If a URN [RFC2141] or other persistent identifier is used, it is
536 RECOMMENDED to express it as an XRI cross-reference as defined in [XRISyntax].
537 Note that for XRI authority resolution, the authority identified by this element is the parent
538 authority (the provider of the current XRD), not the child authority (the target of the
539 current XRD). The latter is identified by the `xrd:XRD/xrd:Service/xrd:ProviderID`
540 element inside a authority resolution service endpoint (see below).

Deleted: For purposes of CanonicalID verification (section 14.3),

Comment [DSR5]: Deleted per revision to section 14.3.2 as suggested by Wil Tan.

541 **xrd:XRD/saml:Assertion**

542 0 or 1 per `xrd:XRD` element. A SAML assertion from the provider of the current XRD
543 that asserts that the information contained in the current XRD is authoritative. Because
544 the assertion is digitally signed and the digital signature encompasses the containing
545 `xrd:XRD` element, it also provides a mechanism for the recipient to detect unauthorized
546 changes since the last time the XRD was published.

547 Note that while a `saml:Issuer` element is required within a `saml:Assertion` element,
548 this specification makes no requirement as to the value of the `saml:Issuer` element. It
549 is up to the XRI community root authority to place restrictions, if any, on the
550 `saml:Issuer` element. A suitable approach is to use an XRI in URI-normal form that
551 identifies the community root authority. See section 9.1.3.

552 **4.2.3 Synonym Elements**

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

557 **xrd:XRD/xrd:LocalID**

558 0 or more per `xrd:XRD` element. Type `xs:anyURI`. Asserts an interchangeable
559 synonym for the value of the `xrd:Query` element. See section 5.2.1 for detailed
560 requirements. Attributes:

- 561
 - `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

562 **xrd:XRD/xrd:EquiVID**

563 0 or more per `xrd:XRD` element. Type `xs:anyURI`. Asserts an absolute identifier for the
564 target resource that is not equivalent to the `CanonicalID` or `CanonicalEquiVID` (see
565 below). See section 5.2.2 for detailed requirements. Attributes:

- 566
 - `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

567 **xrd:XRD/xrd:CanonicalID**

568 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical identifier assigned
569 to the target resource by the authority providing the XRD. See section 5.2.3 for detailed
570 requirements.

571 **xrd:XRD/xrd:CanonicalEquiVID**

572 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical identifier for the
573 target resource assigned by *any* identifier authority. See section 5.2.4 for detailed
574 requirements.

575 **4.2.4 Service Endpoint Descriptor Elements**

576 The next set of elements is used to describe service endpoints—the set of network endpoints
577 advertised in an XRD for performing delegated resolution, obtaining further metadata, or
578 interacting directly with the target resource. Again, because there can be more than one instance
579 of a service endpoint that satisfies a service endpoint selection query, or more than one instance
580 of these elements inside a service descriptor, these elements all accept the global `priority`
581 attribute (section 4.3.3).

582 IMPORTANT: Establishing unambiguous priority is especially important for service endpoints
583 because they are used to control the direction of authority resolution, the order of Redirect and
584 Ref processing, and the prioritization of the final service endpoint URIs selected (if any). See
585 section 4.3.3 for rules and recommendations about usage of the `priority` attribute.

586 Note that to prevent processing conflicts, the XRD schema permits only one of these element
587 types in a service endpoint: `xrd:URI`, `xrd:Redirect`, or `xrd:Ref`.

588 **xrd:XRD/xrd:Service**

589 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.
590 Referred to by the abbreviation *SEP*. Attributes:

- 591 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

592 **xrd:XRD/xrd:Service/xrd:LocalID**

593 0 or more per `xrd:XRD/xrd:Service` element. Identical to the
594 `xrd:XRD/xrd:LocalID` element defined above except this synonym is asserted by the
595 provider of the service and not the parent authority for the XRD. MAY be used to provide
596 one or more identifiers by which the target resource SHOULD be identified in the context
597 of the service endpoint. See section 5.2.1 for detailed requirements. Attributes:

- 598 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

599 **xrd:XRD/xrd:Service/xrd:URI**

600 0 more per `xrd:XRD/xrd:Service` element. Type `xs:anyURI`. Choice between this or
601 the `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
602 elements. If present, it indicates a transport-level URI for accessing the capability
603 described by the parent `Service` element. For the service types defined for XRI resolution
604 in section 3.1.2, this URI MUST be an HTTP or HTTPS URI. Other services may use
605 other transport protocols. Includes an optional `append` attribute that governs construction
606 of the final service endpoint URI as defined in section 13.7. Attributes:

- 607 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

608 **xrd:XRD/xrd:Service/xrd:Redirect**

609 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the
610 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Ref` elements.
611 Identical to the `xrd:XRD/xrd:Redirect` element defined above except processed only
612 in the context of service endpoint selection. See section 12. Attributes:

- 613 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

614 **xrd:XRD/xrd:Service/xrd:Ref**

615 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the
616 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Redirect`
617 elements. Identical to the `xrd:XRD/xrd:Ref` element defined above except processed
618 only in the context of service endpoint selection. See section 12. Attributes:

- 619 • `priority` (type `xs:nonNegativeInteger`). OPTIONAL. See section 4.3.3.

620 4.2.5 Service Endpoint Trust Elements

621 Similar to the XRD trust elements defined above, these elements enable trust to be established in
622 the provider of the service endpoint. These elements are OPTIONAL for generic authority
623 resolution (section 9), but REQUIRED for SAML trusted authority resolution (section 10.2).

624 **xrd:XRD/xrd:Service/xrd:ProviderID**

625 0 or 1 per `xrd:XRD/xrd:Service` element. Identical to the
626 `xrd:XRD/xrd:ProviderID` above, except this identifies the provider of the *described*
627 *service endpoint* instead of the provider of the XRD. For an XRI authority resolution
628 service endpoint, it identifies the *child authority* who will perform resolution of subsequent
629 XRI subsegments. In SAML trusted resolution, when a resolution request is made to the
630 child authority at this service endpoint, the contents of the `xrd:XRD/xrd:ProviderID`
631 element in the response MUST match the content of this element for correlation as
632 defined in section 10.2.5. The same usage MAY apply to other services not defined in
633 this specification. Authors of other specifications employing XRD service endpoints
634 SHOULD define the scope and usage of this element, particularly for trust verification.

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

636 0 or 1 per `xrd:XRD/xrd:Service` element. This element provides the digital signature
637 metadata necessary to validate interaction with the resource identified by the
638 `xrd:XRD/xrd:Service/xrd:ProviderID` (above). In XRI resolution, this element
639 comprises the key distribution method for SAML trusted authority resolution as defined in
640 section 10.2.5. The same usage MAY apply to other services not defined in this
641 specification.

642 4.2.6 Service Endpoint Selection Elements

643 The final set of service endpoint descriptor elements is used in XRI resolution for service endpoint
644 selection. These all include two global attributes used for this purpose: `match` and `select`.

645 **xrd:XRD/xrd:Service/xrd:Type**

646 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI`
647 that identifies the type of capability available at this service endpoint. See section 3.1.2
648 for the resolution service types defined in this specification. If a service endpoint does not
649 include at least one `xrd:Type` element, the service type is effectively described by the
650 type of URI specified in the `xrd:XRD/xrd:Service/xrd:URI` element, i.e., an HTTP
651 URI specifies an HTTP service. See section 13.3.6 for Type element matching rules.
652 Attributes:

- 653 • `match` (type `xs:enumeration`). OPTIONAL. See section 13.3.2.
- 654 • `select` (type `xs:boolean`). OPTIONAL. See section 13.4.2.

655 **xrd:XRD/xrd:Service/xrd:Path**

656 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string
657 meeting the `xri-path` production defined in section 2.2.3 of [XRISyntax]. See section
658 13.3.7 for Path element matching rules. Attributes:

- 659 • `match` (type `xs:enumeration`). OPTIONAL. See section 13.3.2.
- 660 • `select` (type `xs:boolean`). OPTIONAL. See section 13.4.2.

661 **xrd:XRD/xrd:Service/xrd:MediaType**

662 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of
663 content available at this service endpoint. The value of this element MUST be of the form

664 of a media type defined in [RFC2046]. See section 3.3 for the media types used in XRI
665 resolution. See section 13.3.8 for MediaType element matching rules. Attributes:

666 • `match` (type `xs:enumeration`). OPTIONAL. See section 13.3.2.

667 • `select` (type `xs:boolean`). OPTIONAL. See section 13.4.2.

668 The XRD schema (Appendix B) allows other elements and attributes from other namespaces to
669 be added throughout. As described in section 17.1.1, these points of extensibility can be used to
670 deploy new XRI resolution schemes, new service description schemes, or other metadata about
671 the described resource.

672 4.3 XRD Attribute Processing Rules

673 4.3.1 ID Attribute

674 For uses such as SAML trusted resolution (section 10.2) that require unique identification of
675 multiple XRD elements within an XRDS document, the XRD element uses the implicit `xml:id`
676 attribute as defined by the W3C XML ID specification [XMLID]. Note that this attribute is NOT
677 explicitly declared in either the RelaxNG schema in Appendix B or the XML Schema in Appendix
678 C since it is inherently included by the extensibility design of both schemas.

679 If present, the value of this attribute MUST be unique for all elements in the containing XML
680 document. Because an XRI resolver may need to assemble multiple XRDs received from different
681 authority servers into one XRDS document, there MUST be negligible probability that the value of
682 the `xrd:XRD/@xml:id` attribute is not globally unique. For this reason the value of this attribute
683 SHOULD be a UUID as defined by [UUID] prefixed by a single underscore character (“_”) in
684 order to make it a legal *NCName* as required by [XMLID]. However, the value of this attribute
685 MAY be generated by any algorithm that fulfills the same requirements of global uniqueness and
686 *NCName* conformance.

687 Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their
688 XML document order MUST match the order in which they were resolved (see section 9.1.2).
689 Also, if Redirect or Ref processing requires the same XRD to be included in an XRDS document
690 twice (via a nested XRDS document), that XRD MUST reference the previous instance using the
691 `xrd:XRD/@idref` attribute as defined in section 12.5.

692 4.3.2 Version Attribute

693 Unlike the XRDS element, which is not intended to be versioned, the `xrd:XRD` element has the
694 optional attribute `xrd:XRD/@version`. Use of this attribute is REQUIRED for XRI resolution.
695 The value of this attribute MUST be the exact numeric version value of the XRI Resolution
696 specification to which the containing XRD element conforms. See sections 3.1.1 and 17.2.1.

697 General rules about versioning of the XRI resolution protocol are defined in section 17.2. Specific
698 rules for processing the XRD version attribute are specified in section 17.2.4.

699 4.3.3 Priority Attribute

700 Certain XRD elements involved in the XRI resolution process (`xrd:Redirect`, `xrd:Ref`,
701 `xrd:Service`, and `xrd:URI`) may be present multiple times in an XRDS document to enable
702 delegation, provide redundancy, expose differing capabilities, or other purposes. In this case XRD
703 authors SHOULD use the global `priority` attribute to prioritize selection of these element
704 instances. Like the priority attribute of DNS records, this attribute accepts a non-negative integer
705 value.

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

- 709 1. The consuming application SHOULD select the element instance with the lowest numeric
710 value of the `priority` attribute. For example, an element with `priority` attribute value
711 of “10” should be selected before an element with a `priority` attribute value of “11”,
712 and an element with `priority` attribute value of “11” should be selected before an
713 element with a `priority` attribute value of “25”. Zero is the highest `priority` attribute
714 value. Null is the lowest `priority` attribute value—it is the equivalent of a value of
715 infinity. It is RECOMMENDED to use a large finite value (100 or more) rather than a null
716 value.
- 717 2. If an element has no `priority` attribute, its `priority` attribute value is considered to
718 be null, i.e., the lowest possible priority value. Rather than omitting a `priority` attribute,
719 it is RECOMMENDED that XRI authorities follow the standard practice in DNS and set
720 the default `priority` attribute value to “10”.
- 721 3. If two or more instances of the same element type have identical `priority` attribute
722 values (including the null value), the consuming application SHOULD select one of the
723 instances at random. This consuming application SHOULD NOT simply choose the first
724 instance that appears in XML document order.

725 **IMPORTANT:** It is vital that implementers observe the preceding rule in order to support
726 intentional redundancy or load balancing semantics. At the same time, it is vital that XRDS
727 authors understand that this rule can result in non-deterministic behavior if two or more of the
728 same type of synonym elements or service endpoint elements are included with the same priority
729 in an XRD but are NOT intended for redundancy or load balancing.

- 730 4. An element selected according to these rules is referred to in this specification as *the*
731 *highest priority element*. If this element is subsequently disqualified from the set of
732 qualified elements, the next element selected according to these rules is referred to as
733 *the next highest priority element*. If a resolution operation specifying selection of the
734 highest priority element fails, the resolver SHOULD attempt to select the next highest
735 priority element unless otherwise specified. This process SHOULD be continued for all
736 other instances of the qualified elements until success is achieved or all instances are
737 exhausted.

738 4.4 XRI and IRI Encoding Requirements

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

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

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

752 5 XRD Synonym Elements

753 XRDS architecture includes support for *synonyms*—XRI, IRIs, or URIs that are not character-for-
754 character equivalent, but which identify the same target resource (in the same context, or across
755 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

756 Table 7: The four XRD synonym elements.

757 This section specifies the normative rules for usage of each XRD synonym element.

758 5.1 Query Identifiers

759 Each XRI synonym element asserts a synonym for the *query identifier*. This is the identifier
760 resolved to obtain the XRDS document containing the XRD asserting the synonym. A *fully-*
761 *qualified query identifier* may be either:

- 762 1. A valid absolute HTTP(S) URI that does not contain an XRI.
- 763 2. A valid absolute XRI, either in a standard XRI form as defined in **[XRISyntax]**, or
764 encoded in an HTTP(S) URI (called an *HXRI*) as specified in section 11.2.

765 5.1.1 HTTP(S) URI Query Identifiers

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

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

772 5.1.2 XRI Query Identifiers

773 If the fully-qualified query identifier is an absolute XRI, the XRDS document to which it resolves
774 (via the protocol specified in section 9.1.2) MAY contain multiple XRDs, each XRD corresponding
775 to one subsegment of the authority component of the XRI. Each XRD SHOULD include an
776 `xrd:Query` element that echos back the XRI subsegment described by this XRD. This is called
777 the *local query identifier*, because it represents just one subsegment of the fully-qualified query
778 identifier.

779 At any point in the XRI resolution chain, the combination of the community root authority XRI
780 (section 9.1.3) plus all local query identifiers resolved in all XRDs up to that point is called the
781 *current fully-qualified query identifier*. When the resolution chain is complete, the current fully-
782 qualified query identifier is equal to the starting fully-qualified query identifier.

783 In each XRD in the resolution chain, the LocalID element asserts a synonym for the local query
784 identifier, and the EquivID, CanonicalID, and CanonicalEquivID elements assert a synonym for
785 the current fully-qualified query identifier.

786 5.2 Synonym Elements

787 5.2.1 LocalID

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

791 At the XRD level, the value of the `xrd:XRD/xrd:LocalID` element asserts a synonym that is
792 interchangeable with the contents of the `xrd:Query` element in the XRD. This means that
793 resolution of a LocalID in the context of the same parent authority using the same resolution
794 query parameters as the current query MUST result in an equivalent XRD as defined in section
795 5.4. It also means an XRI resolver MAY use a LocalID as an alternate key for the XRD in its
796 cache (see section 16.4.2).

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

802 At the SEP level, the `xrd:XRD/xrd:Service/xrd:LocalID` element MAY be used to express
803 either a local or global identifier for the target resource in the context of the specific service being
804 described. If present, consuming applications SHOULD use the value of the highest priority
805 instance of the `xrd:XRD/xrd:Service/xrd:LocalID` element to identify the target resource
806 in the context of this service endpoint. If not present, consuming applications SHOULD select a
807 synonym as defined in section 5.6.

808 SPECIAL SECURITY CONSIDERATIONS: A parent authority SHOULD NOT permit a child
809 authority to edit a LocalID value in an XRD without authenticating the child authority and verifying
810 that the child authority is authorized to use this LocalID value either at the XRD level and/or the
811 SEP level.

812 5.2.2 EquivID

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

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

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

- 819 1. When it is REQUIRED as a backpointer to verify another EquivID element in a different
820 XRD as specified in section 14.2.
- 821 2. When it is REQUIRED as a backpointer to verify a CanonicalEquivID element as
822 specified in section 14.3.3.

823 SPECIAL SECURITY CONSIDERATIONS: An EquivID synonym SHOULD NOT be trusted
824 unless it is verified. This function is not performed automatically by XRI resolvers but may be
825 easily performed by consuming applications using one additional XRI resolution call as specified
826 in section 14.2. A parent authority SHOULD NOT permit a child authority to edit the EquivID value
827 in an XRD without authenticating the child authority and verifying that the child authority is

828 authorized to use this EquivID value. A parent authority SHOULD NOT assert an EquivID
829 element if the identifier authority to whom it points is not authorized to make a CanonicalEquivID
830 assertion.

831 5.2.3 CanonicalID

832 The purpose of the `xrd:CanonicalID` element is to assert the canonical identifier assigned by
833 the parent authority to the target resource described by an XRD. It plays a special role in XRD
834 synonym architecture because it is the ultimate test of XRD equivalence as defined in section 5.4.
835 A CanonicalID MUST meet all the requirements of an EquivID plus the following:

- 836 1. It MUST be an identifier for which the parent authority is the final authority. This means
837 that resolution of a CanonicalID using the same resolution query parameters as the
838 current query MUST result in an equivalent XRD as defined in section 5.4.
- 839 2. If the CanonicalID is any XRI except a community root authority XRI (section 9.1.3), it
840 MUST consist of the parent authority's CanonicalID plus one additional subsegment. (In
841 XRI resolution the parent authority's CanonicalID is always in the immediately preceding
842 XRD in the same XRDS document, not in a nested XRDS document produced as a result
843 of Redirect and Ref processing as defined in section 12.5.) For example, if the
844 CanonicalID asserted for a target resource is `@!1!2!3`, then the CanonicalID for the
845 parent authority must be `@!1!2`. See section 14.3.2 for details.
- 846 3. Once assigned, a parent authority SHOULD NEVER: a) change or reassign a
847 CanonicalID value, or b) stop asserting a CanonicalID element in an XRD in which it has
848 been asserted. For this reason, it is STRONGLY RECOMMENDED to use a persistent
849 identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

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

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

856 SPECIAL SECURITY CONSIDERATIONS: A CanonicalID synonym SHOULD NOT be trusted
857 unless it is verified. CanonicalID verification is performed automatically during resolution by an
858 XRI resolver unless this function is explicitly turned off; see section 14. A parent authority
859 SHOULD NOT permit a child authority to edit the CanonicalID value in an XRD without
860 authenticating the child authority and verifying that the child authority is authorized to use this
861 CanonicalID value.

862 5.2.4 CanonicalEquivID

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

- 866 1. In order for the value of the `xrd:CanonicalEquivID` element to be verified: a) the
867 XRD in which it appears MUST include a CanonicalID that can be verified as specified in
868 section 14.2, and b) the XRD to which it resolves MUST meet the rules specified in
869 section 14.3.3. In particular, those rules require that the CanonicalID of that XRD match
870 the asserted CanonicalEquivID.
- 871 2. For the same reasons as with a CanonicalID, it is STRONGLY RECOMMENDED to use
872 a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

873 3. Although the CanonicalEquivID associated with a CanonicalID MAY change over time, at
874 any one point in time, every XRD from the same parent authority that asserts the same
875 CanonicalID value MUST assert the same CanonicalEquivID value if the XRD includes a
876 CanonicalEquivID element.

877 As a best practice, a parent authority SHOULD publish a CanonicalEquivID in an XRD if
878 consuming applications SHOULD be able to persistently identify the target resource using this
879 identifier in other contexts. Also, a CanonicalEquivID value SHOULD change very infrequently, if
880 at all.

881 SPECIAL SECURITY CONSIDERATIONS: A CanonicalEquivID synonym SHOULD NOT be
882 trusted unless it is verified. Verification of the value of the CanonicalEquivID element in the final
883 XRD in an XRDS document is performed automatically during resolution by an XRI resolver
884 unless this function is explicitly turned off; see section 14. A parent authority SHOULD NOT
885 permit a child authority to edit the CanonicalEquivID value in an XRD without authenticating the
886 child authority and verifying that the child authority is authorized to use this CanonicalEquivID
887 value.

888 5.3 Redirect and Ref Elements

889 While similar in some ways to synonym elements, the `xrd:Redirect` and `xrd:Ref` elements
890 MUST NOT be used to assert a synonym. Instead their purpose is to assert that a different XRDS
891 document is authorized to serve as an equally valid descriptor of the target resource. These
892 elements enable separation of synonym assertion semantics vs. distributed XRDS document
893 authorization semantics.

894 In the same way as a LocalID, both a Redirect and a Ref may be used in an XRD at either the
895 XRD level (as a child of the root `xrd:XRD` element) and at the SEP level (as a child of the root
896 `xrd:XRD/xrd:Service` element). The complete rules for Redirect and Ref processing in XRI
897 resolution are specified in section 12.

898 If two independent resources are later merged into the same resource, e.g., two businesses are
899 merged into one, the use of an EquivID, CanonicalID, or CanonicalEquivID element SHOULD be
900 combined with the use of a Redirect or Ref element to provide the semantics of BOTH identifier
901 synonymity and XRDS authorization.

902 SPECIAL SECURITY CONSIDERATIONS: A parent authority SHOULD NOT permit a child
903 authority to edit a Redirect or Ref value in an XRD without authenticating the child authority and
904 verifying that the child authority is authorized to use this Redirect or Ref value at either the XRD
905 level and/or the SEP level.

906 5.4 XRD Equivalence

907 LocalID and CanonicalID synonyms are required to resolve to an XRD that is equivalent to the
908 XRD in which the synonym is asserted. Two XRDS MUST be considered equivalent if they meet
909 the following rules:

- 910 1. Both XRDS contain a CanonicalID element.
- 911 2. The values of these CanonicalID elements are equivalent according to the equivalence
912 rules of the applicable identifier scheme. Note that these identifiers MUST be in URI-
913 normal form as specified in section 4.4. In addition, if the CanonicalID values are
914 HTTP(S) URIs, fragments MUST be considered significant in comparison.

915 In addition, while not strictly required for XRD equivalence, section 5.2.4 REQUIRES that two
916 equivalent XRDS issued at the same point in time assert the same CanonicalEquivID value if they
917 both contain a CanonicalEquivID element. It is RECOMMENDED that all other elements in the
918 XRD that are not relative to a specific resolution request also be equivalent.

919 5.5 Synonym Verification

920 For security purposes, it is STRONGLY RECOMMENDED that a consuming application not rely
921 on EquivID, CanonicalID, or CanonicalEquivID synonyms unless they are verified as specified in
922 section 14.

923 5.6 Synonym Selection

924 It is out of the scope of this specification to specify policies consuming applications should use to
925 select their desired synonym(s) to identify a target resource. However, the following are
926 RECOMMENDED best practices:

- 927 • Only select a verified synonym (see above).
- 928 • Select a persistent synonym, particularly if a long term or immutable reference is required. If
929 a persistent synonym is present, other reassignable synonyms (including the current fully-
930 qualified query identifier) SHOULD be treated only as temporary identifiers.
- 931 • Select a CanonicalID if present, verified, and persistent. This identifier SHOULD be used
932 whenever referencing the target resource in the context of the parent authority issuing the
933 CanonicalID.
- 934 • If possible, *also* select a CanonicalEquivID if present, verified, and persistent. This identifier
935 SHOULD be used as a reference to the target resource in any context other than that of the
936 parent authority.
- 937 • When selecting a synonym to use in the context of a specific service endpoint, follow the
938 recommendations for use of the `xrd:XRD/xrd:Service/xrd:LocalID` element as
939 specified in section 5.2.1.

940 6 Discovering an XRDS Document from an 941 HTTP(S) URI

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

946 6.1 Overview

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

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

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

955 6.2 HEAD Protocol

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

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

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

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

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

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

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

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

973 6.3 GET Protocol

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

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

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

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

984 If the response is only HTTP(S) response headers as defined in section 6.2, or if in addition to
985 these response headers it includes any document other than the two document types defined in
986 the third and fourth options above, the protocol MUST proceed as defined in section 6.2, *except*
987 *that there is no fallback to this section if that protocol fails.*

988 If the response is only an HTML document as defined in the third option above, the value of the
989 <meta> element with an http-equiv attribute equal to X-XRDS-Location MUST be an
990 HTTP(S) URI which gives the location of an XRDS document describing the target resource. If
991 this HTTP(S) URI is identical to the starting HTTP(S) URI, this is a loop and the protocol fails.
992 Otherwise, the XRDS client MUST request the XRDS document from this URI using an HTTP(S)
993 GET. This request SHOULD include an Accept header specifying the content type
994 application/xrds+xml.

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

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

1000 In all cases the HTTP(S) status messages and error codes defined in **[RFC2616]** apply.

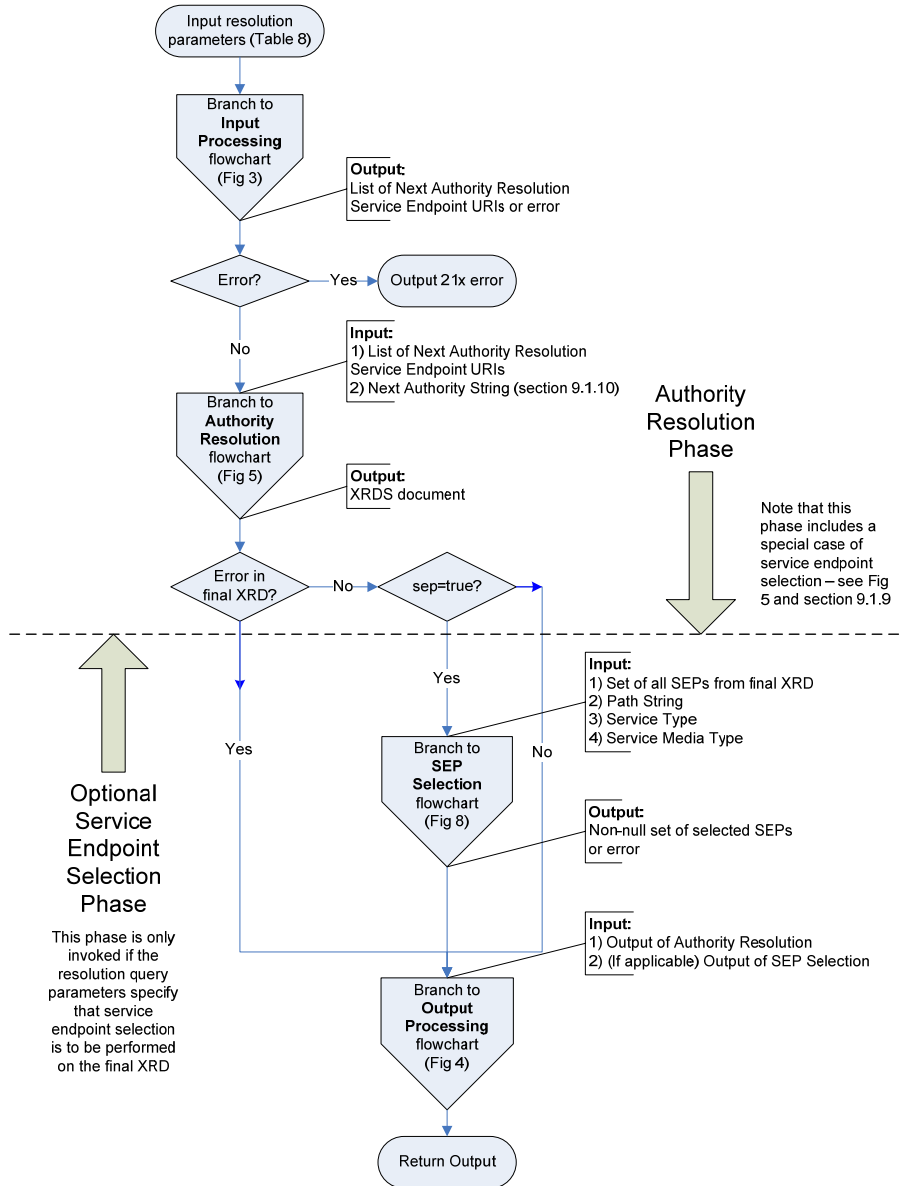
1001 Note: If the XRDS server supports content negotiation, the response SHOULD include a Vary:
1002 header to allow caches to properly interpret future requests. This header SHOULD be present
1003 even in the case where the HTML page is returned (instead of an XRDS document).

1004

7 XRI Resolution Flow

1005
1006
1007
1008

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). Figure 2 is a top-level flowchart of this function highlighting the two major phases: *authority resolution* followed by *optional service endpoint selection*.



1009

1010

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

1011 Branches of this top-level flowchart are used throughout the specification to provide a logical
1012 overview of key components of XRI resolution. The branch flowcharts include:

- 1013 • Figure 3: Input processing (section 8.1).
- 1014 • Figure 4: Output processing (section 8.2).
- 1015 • **Figure 5: Authority resolution (section 9).**
- 1016 • Figure 6: XRDS requests (section 9.1.3).
- 1017 • **Figure 7: Redirect and Ref processing (section 12).**
- 1018 • **Figure 8: Service endpoint selection (section 13).**
- 1019 • Figure 9: Service endpoint selection logic (section 13.2).

1020 **IMPORTANT:** In all cases the flowcharts are informative and the specification text is normative.
1021 However, the flowcharts are recommended as an aid in reading the specification. In particular,
1022 those highlighted in **bold** above illustrate the recursive calls for authority resolution and service
1023 endpoint selection used during Redirect and Ref processing (section 12). Implementers should
1024 pay special attention to these calls and the guidance in section 12.6, *Recursion and Backtracking*.

1025

8 Inputs and Outputs

1026
1027
1028
1029

This section defines the logical inputs and outputs of XRI resolution together with their processing rules. 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 11. For purposes of illustration, a binding to a non-normative, language-neutral API is suggested in Appendix F.

1030

8.1 Inputs

1031
1032
1033
1034

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

Logical Input Parameter Name	Type	Required/Optional	Default	Resolution Phase	Section
QXRI (query XRI) including Authority String, Path String, and Query String	xs:anyURI	Required	N/A	Authority Resolution (except Path String which is used in Service Endpoint Selection)	8.1.1
Resolution Output Format	xs:string (media type)	Optional	Null	Authority Resolution	8.1.2
Service Type	xs:anyURI	Optional	Null	Service Endpoint Selection	8.1.3
Service Media Type	xs:string (media type)	Optional	Null	Service Endpoint Selection	8.1.4

1035

Table 8: Input parameters for XRI resolution.

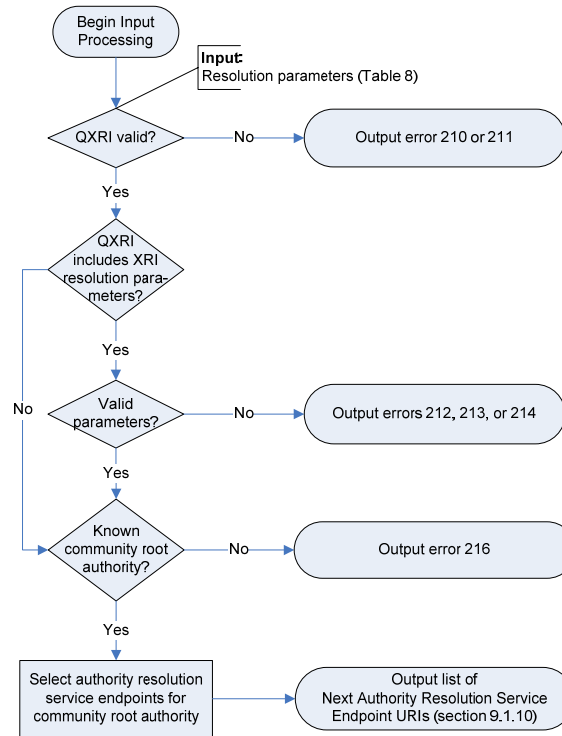
1036
1037

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

1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048

1. The presence of an input parameter, subparameter, or XRD element with an empty value MUST be treated as equivalent to the absence of that input parameter, subparameter, or XRD element. (Note that this rule does not apply to XRD attributes.)
2. From a programmatic standpoint, both conditions above MUST be considered as equivalent to setting the value of that parameter, subparameter, or element to null.
3. In an XRD element, an attribute with an empty value is an error and MUST NOT be interpreted as the default value or any other value of that attribute.
4. As required by **[XMLSchema2]**, for all Boolean subparameters: a) the string values `true` and `false` MUST be considered case-insensitive (lowercase is RECOMMENDED), b) the values `true` and `1` MUST be considered equivalent, b) the values `false` and `0` MUST be considered equivalent.

1049 Figure 3 is a flowchart (non-normative) illustrating the processing of input parameters.



1050

1051 *Figure 3: Input processing flowchart.*

1052 The following sections specify additional validation and usage requirements that apply to
1053 particular input parameters.

1054 **8.1.1 QXRI (Authority String, Path String, and Query String)**

1055 The QXRI (query XRI) is the only REQUIRED input parameter. Per [XRISyntax], a QXRI consists
1056 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 component 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 "/", "?", "#", whitespace, 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.

1057 *Table 9: Subparameters of the QXRI input parameter.*

1058 The fourth possible component of a QXRI—a fragment—is by definition resolved locally relative
1059 to the target resource identified by the combination of the Authority, Path, and Query
1060 components, and as such does not play a role in XRI resolution.

1061 Following are the constraints on the value of the QXRI parameter.

- 1062 1. It MUST be a valid absolute XRI according to the ABNF defined in [XRISyntax]. To
1063 resolve a relative XRI reference, it must be converted into an absolute XRI using the
1064 procedure defined in section 2.4 of [XRISyntax].
- 1065 2. For authority or proxy resolution as defined in this specification, the QXRI MUST be in
1066 URI-normal form as defined in section 2.3.1 of [XRISyntax]. A local resolver API MAY
1067 support the input of other XRI forms but SHOULD document the normal form(s) it
1068 supports and its normalization policies.
- 1069 3. When a QXRI is included as part of an HXRI (section 11.2) for XRI proxy resolution, the
1070 QXRI MUST be normalized as specified in section 11.2, and all HXRI query parameters
1071 MUST follow the encoding rules specified in sections 11.3 and 11.4.

1072 **8.1.2 Resolution Output Format**

1073 The Resolution Output Format is an OPTIONAL parameter that, together with its subparameters,
1074 is used to specify:

- 1075 • The media type for the resolution response.
- 1076 • Whether generic or trusted resolution must be used by the resolver.
- 1077 • Whether Refs should be followed during resolution.
- 1078 • Whether CanonicalID verification should not be performed during resolution.
- 1079 • Whether service endpoint selection should be performed on the final XRD.

- 1080 • Whether default matches should be ignored during service endpoint selection.
- 1081 • Whether URIs should automatically be constructed in the final XRD.

1082 Following are the normative requirements for the use of this parameter.

- 1083 1. The Resolution Output Format MUST be one of the values specified in Table 5 and MAY
1084 include any of the subparameters specified in Table 6.
- 1085 2. If the value of the `https` subparameter is TRUE, the resolver MUST use the HTTPS
1086 trusted authority resolution protocol specified in section 10.1 (or return an error indicating
1087 this is not supported).
- 1088 3. If the value of the `saml` subparameter is TRUE, the resolver MUST use the SAML trusted
1089 authority resolution protocol specified in section 10.2 (or return an error indicating this is
1090 not supported).
- 1091 4. If the value of both the `https` and `saml` subparameters are TRUE, the resolver MUST
1092 use the HTTPS+SAML trusted authority resolution protocol specified in section 10.3 (or
1093 return an error indicating this is not supported).
- 1094 5. If the value of the `cid` subparameter is TRUE or null, or if the parameter is absent, the
1095 resolver MUST perform CanonicalID verification as specified in section 14.3. If the value
1096 of the `cid` subparameter is FALSE, the resolver MUST NOT perform CanonicalID
1097 verification.
- 1098 6. If the value of the `refs` subparameter is TRUE or null, or if the parameter is absent, the
1099 resolver MUST perform Ref processing as specified in section 12. If the value of the
1100 `refs` subparameter is FALSE, the resolver MUST NOT perform Ref processing and
1101 must return an error if a Ref is encountered as specified in section 12.
- 1102 7. If the value of the `sep` subparameter is TRUE, the resolver MUST perform service
1103 endpoint selection on the final XRD (even if the values of all service endpoint selection
1104 parameters are null). If the value of the `sep` subparameter is FALSE or null, or if the
1105 parameter is absent, the resolver MUST NOT perform service endpoint selection on the
1106 final XRD unless it is required to produce a URI List or HTTP(S) redirect. See section 8.2.
- 1107 8. If the value of the `nodefault_t`, `nodefault_p`, or `nodefault_m` subparameter is
1108 TRUE, the resolver MUST ignore default matches on the corresponding service endpoint
1109 selection element categories as specified in section 13.3.2.
- 1110 9. If the value of the `uric` subparameter is TRUE, the resolver MUST perform service
1111 endpoint URI construction as specified in section 13.7.1. If the value of the `uric`
1112 subparameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT
1113 perform service endpoint URI construction.

1114 Future versions of this specification, or other specifications for XRI resolution, MAY use other
1115 values for Resolution Output Format or its subparameters.

1116 8.1.3 Service Type

1117 The Service Type is an OPTIONAL value of type `xs:anyURI` used to request a specific type of
1118 service in the service endpoint selection phase (section 11). The value of this parameter MUST
1119 be a valid absolute XRI, IRI, or URI in URI-normal form as defined by **[XRISyntax]**. (Note that
1120 URI-normal form is required so this parameter may be passed to a proxy resolver in a QXRI
1121 query parameter as defined in section 11.) The Service Type values defined for XRI resolution
1122 services are specified in section 3.1.2. The rules for matching the value of the Service Type
1123 parameter to the value of the `xrd:XRD/xrd:Service/xrd:Type` element are specified in
1124 section 13.3.6.

1125 **8.1.4 Service Media Type**

1126 The Service Media Type is an OPTIONAL string used to request a specific media type in the
1127 service endpoint selection phase (section 11). The value of this parameter MUST be a valid
1128 media type as defined by [RFC2046]. The Service Media Type values defined for XRI resolution
1129 services are specified in section 3.3. The rules for matching the value of the Service Media Type
1130 parameter to the value of the `xrd:XRD/xrd:Service/xrd:MediaType` element are specified
1131 in section 13.3.8.

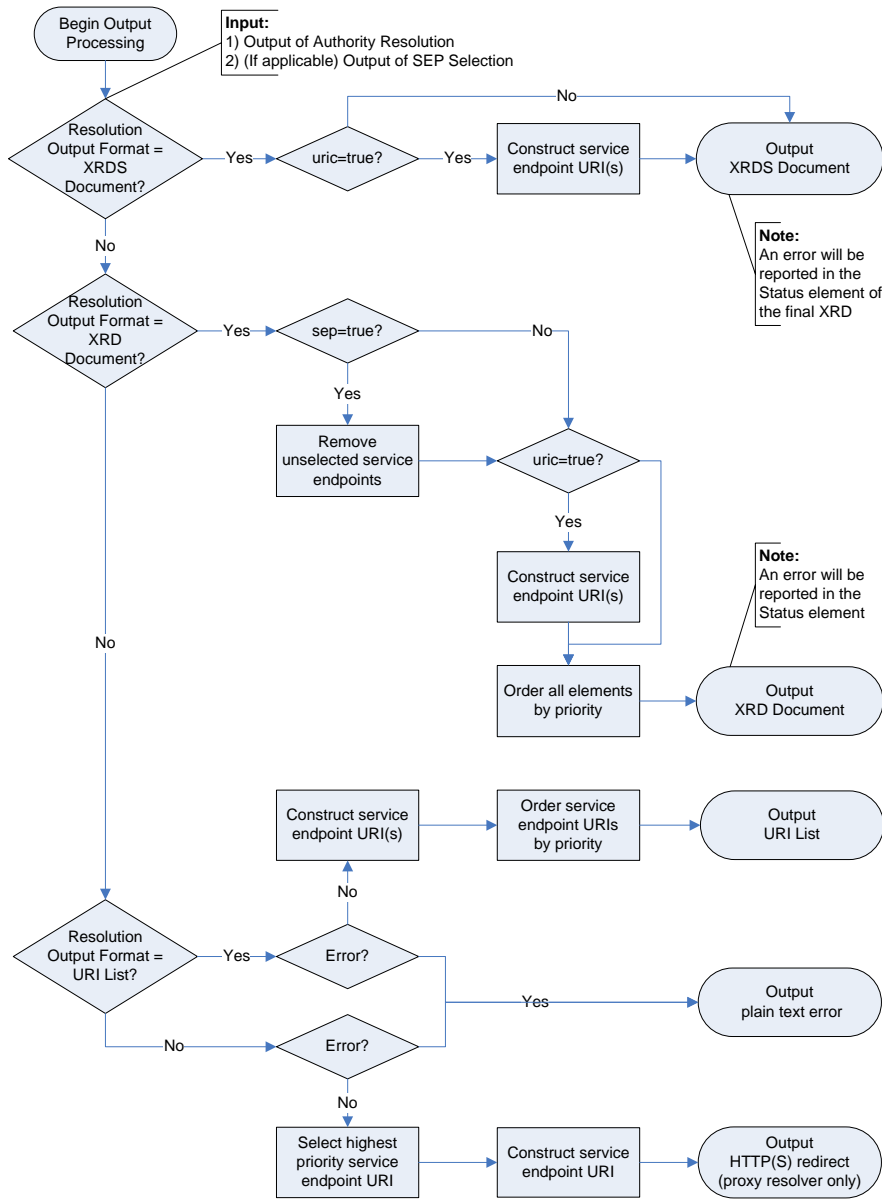
1132 **8.2 Outputs**

1133 Table 10 summarizes the logical outputs of XRI resolution. Note that these are defined in terms of
1134 media types returned by authority servers and proxy resolvers. A local resolver API MAY
1135 implement other representations of these media types.

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 Element	application/xrd+xml	application/xrd+xml;sep=true
URI List	N/A	text/uri-list
HTTP(S) Redirect	N/A	<i>null</i>

1136 *Table 10: Outputs of XRI resolution.*

1137 Figure 4 is a flowchart illustrating the process of producing these output formats once the auth-
 1138 ority resolution and optional service endpoint selection phases are complete. Note that in the first
 1139 two output options, errors are reported directly in the XRDS, so no special error format is needed.



1140
 1141 *Figure 4: Output processing flowchart.*

1142 The following sections provide additional construction and validation requirements.

1143 8.2.1 XRDS Document

1144 If the value of the Resolution Output Format parameter is `application/xrds+xml`, the
1145 following rules apply.

- 1146 1. The output MUST be a valid XRDS document according to the schema defined in
1147 Appendix B.
- 1148 2. The XRDS document MUST contain an ordered list of `xrd:XRD` elements—one for each
1149 authority subsegment successfully resolved by the resolver client. This list MUST appear
1150 in the same order as the corresponding subsegments in the Authority String.
- 1151 3. Each of the contained XRD elements must be a valid XRD element according to the
1152 schema defined in Appendix B.
- 1153 4. The XRD elements MUST conform to the additional requirements in section 4.
- 1154 5. If the value of the `saml` subparameter of the Resolution Output Format is TRUE, the
1155 XRD elements MUST conform to the additional requirements in section 10.2.
- 1156 6. If Redirect or Ref processing is necessary during the authority resolution or service
1157 endpoint selection process, it MUST result in a valid nested XRDS document as defined
1158 in section 12.
- 1159 7. If the value of the `sep` subparameter is TRUE, service endpoint selection MUST be
1160 performed as defined in section 13, even if the values of all three service endpoint
1161 selection input parameters (Service Type, Path String, and Service Media Type) are null.

1162 **IMPORTANT:** No filtering of the final XRD is performed when returning an XRDS document.
1163 Filtering is only performed when the requested Resolution Output Format is an XRD element –
1164 see the next section.

- 1165 8. If the value of the `cid` subparameter is TRUE, synonym verification MUST be reported
1166 using the `xrd:Status` element of each XRD in the XRDS document as defined in
1167 section 14.
- 1168 9. If the output is an error, this error MUST be returned using the `xrd:Status` element of
1169 the final XRD in the XRDS document as defined in section 15.

1170 8.2.2 XRD Element

1171 If the value of the Resolution Output Format parameter is `application/xrd+xml`, the following
1172 rules apply.

- 1173 1. The output MUST be a valid XRD element according to the schema defined in Appendix
1174 B.
- 1175 2. The XRD elements MUST conform to the additional requirements in section 4.
- 1176 3. If the value of the `saml` subparameter of the Resolution Output Format is TRUE, the
1177 XRD element MUST conform to the additional requirements in section 10.2.
- 1178 4. If the value of the `sep` subparameter is FALSE or null, or if this parameter is absent, the
1179 XRD MUST be the final XRD in the XRDS document produced as a result of authority
1180 resolution. Service endpoint selection or any other filtering of the XRD element MUST
1181 NOT be performed.
- 1182 5. If the value of the `sep` subparameter is TRUE, service endpoint selection MUST be
1183 performed as defined in section 13, even if the values of all service endpoint selection
1184 input parameters are null.
- 1185 6. If service endpoint selection is performed, the only `xrd:Service` elements in the XRD
1186 element MUST be those selected according to the rules specified in section 13. If no
1187 service endpoints were selected by those rules, no `xrd:Service` elements will be

1188 present. In addition, all elements within the XRD element that are subject to the global
1189 `priority` attribute (even if the attribute is absent or null) MUST be returned in order of
1190 highest to lowest priority as defined in section 4.3.3.

1191 **IMPORTANT:** Any other filtering of the XRD element MUST NOT be performed. Note that this
1192 means that if the XRD element includes a SAML signature element as defined in section 10.2,
1193 this element is still returned inside the XRD element even though it may not be able to be verified
1194 by a consuming application.

- 1195 7. If the value of the `cid` subparameter is TRUE, synonym verification MUST be reported
1196 using the `xrd:Status` element of each XRD in the XRDS document as defined in
1197 section 14.
- 1198 8. If the output is an error, this error MUST be returned using the `xrd:Status` element as
1199 defined in section 15.

1200 8.2.3 URI List

1201 If the value of the Resolution Output Format parameter is `text/uri-list`, the following rules
1202 apply.

- 1203 1. For this output, service endpoint selection is REQUIRED, even if the values of all service
1204 endpoint selection input parameters are null.
- 1205 2. If authority resolution and service endpoint selection are both successful, the output
1206 MUST be a valid URI List as defined by section 5 of [RFC2483].
- 1207 3. If, after applying the service endpoint selection rules, more than one service endpoint is
1208 selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as
1209 defined in section 4.3.3.
- 1210 4. If the final selected `xrd:XRD/xrd:Service` element contains a
1211 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
1212 element, Redirect and Ref processing MUST be performed as described in section 12.
1213 This rule applies iteratively to each new XRDS document resolved.
- 1214 5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)
1215 MUST be constructed as defined in section 13.7.1.
- 1216 6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`
1217 elements within the selected `xrd:Service` element as defined in section 4.3.3. When
1218 two or more of the source `xrd:URI` elements have equal priority, their constructed URIs
1219 SHOULD be returned in random order.

1220 **IMPORTANT:** Any other filtering of the URI list MUST NOT be performed.

- 1221 7. If the output is an error, it MUST be returned with the content type `text/plain` as
1222 defined in section 15.

1223 8.2.4 HTTP(S) Redirect

1224 In XRI proxy resolution, the Resolution Output Format parameter may be null. In this case the
1225 output of a proxy resolver is an HTTP(S) redirect as defined in section 11.7.

1226

9 Generic Authority Resolution Service

1227
1228
1229
1230

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

1231
1232
1233
1234
1235

XRI authorities and IRI authorities have different syntactic structures, partially due to the higher level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved to XRDS documents one subsegment at a time as specified in section 9.1. IRI authorities, since they are based on DNS names or IP addresses, are resolved into an XRDS document through a special HTTP(S) request using the entire IRI authority component as specified in section 9.1.11.

1236

9.1 XRI Authority Resolution

1237

9.1.1 Service Type and Service Media Type

1238

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

Service Type	Service Media Type	Subparameters
xri://\$res*auth*(\$v*2.0)	application/xrds+xml	OPTIONAL (see important note below)

1239

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

1240
1241

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

1242
1243
1244
1245
1246

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

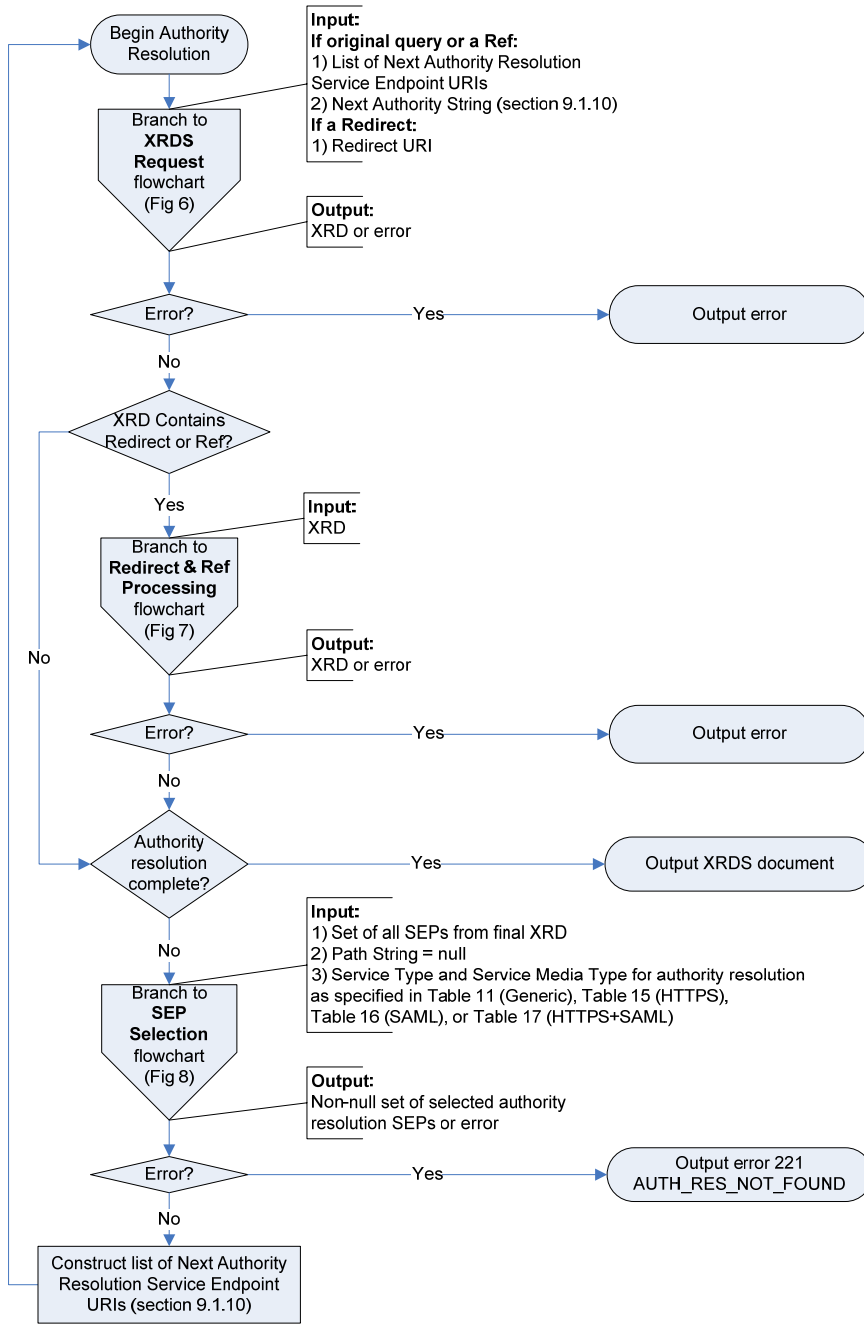
1247
1248
1249
1250
1251
1252

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

1253 **9.1.2 Protocol**

1254 Figure 5 (non-normative) illustrates the overall logical flow of generic authority resolution.

Comment [DSR6]: This flowchart revised for improved clarity about Redirect URI as an input and recording of XRD and XRDS documents.



1255
1256 *Figure 5: Authority resolution flowchart.*

- 1257 Following are the normative requirements for behavior of an XRI resolver and an XRI authority
1258 server when performing generic XRI authority resolution:
- 1259 1. Each request for an XRDS document using HTTP(S) MUST conform to the requirements
1260 in section 9.1.3.
 - 1261 2. For errors in XRDS document resolution requests, a resolver MUST implement failover
1262 handling as specified in section 9.1.4.
 - 1263 3. The resolver MUST be preconfigured with or have a means of obtaining the XRDS
1264 document describing the community root authority for the XRI to be resolved as defined
1265 in section 9.1.5.
 - 1266 4. The resolver MAY obtain the XRDS document describing the community root authority by
1267 requesting a self-describing XRDS document as defined in section 9.1.6.
 - 1268 5. Resolution of each subsegment in the Authority String after the community root
1269 subsegment MUST proceed in subsegment order (left-to-right) using fully qualified
1270 subsegment values as defined in section 9.1.7.
 - 1271 6. Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as
1272 defined in section 9.1.8.
 - 1273 7. For each iteration of the authority resolution process, the next authority resolution service
1274 endpoint MUST be selected as specified in section 9.1.9.
 - 1275 8. For each iteration of the authority resolution process, an HTTP(S) URI (called the Next
1276 Authority URI) MUST be constructed according to the algorithm specified in section
1277 9.1.10.
 - 1278 9. A resolver MAY request that a recursing authority server perform resolution of multiple
1279 subsegments as defined in section 9.1.11.
 - 1280 10. For each iteration of the authority resolution process, a resolver MUST perform Redirect
1281 and Ref processing as specified in section 12. Note that if Redirect and Ref processing is
1282 successful, it will result in a nested XRDS document as specified in section 12.5.

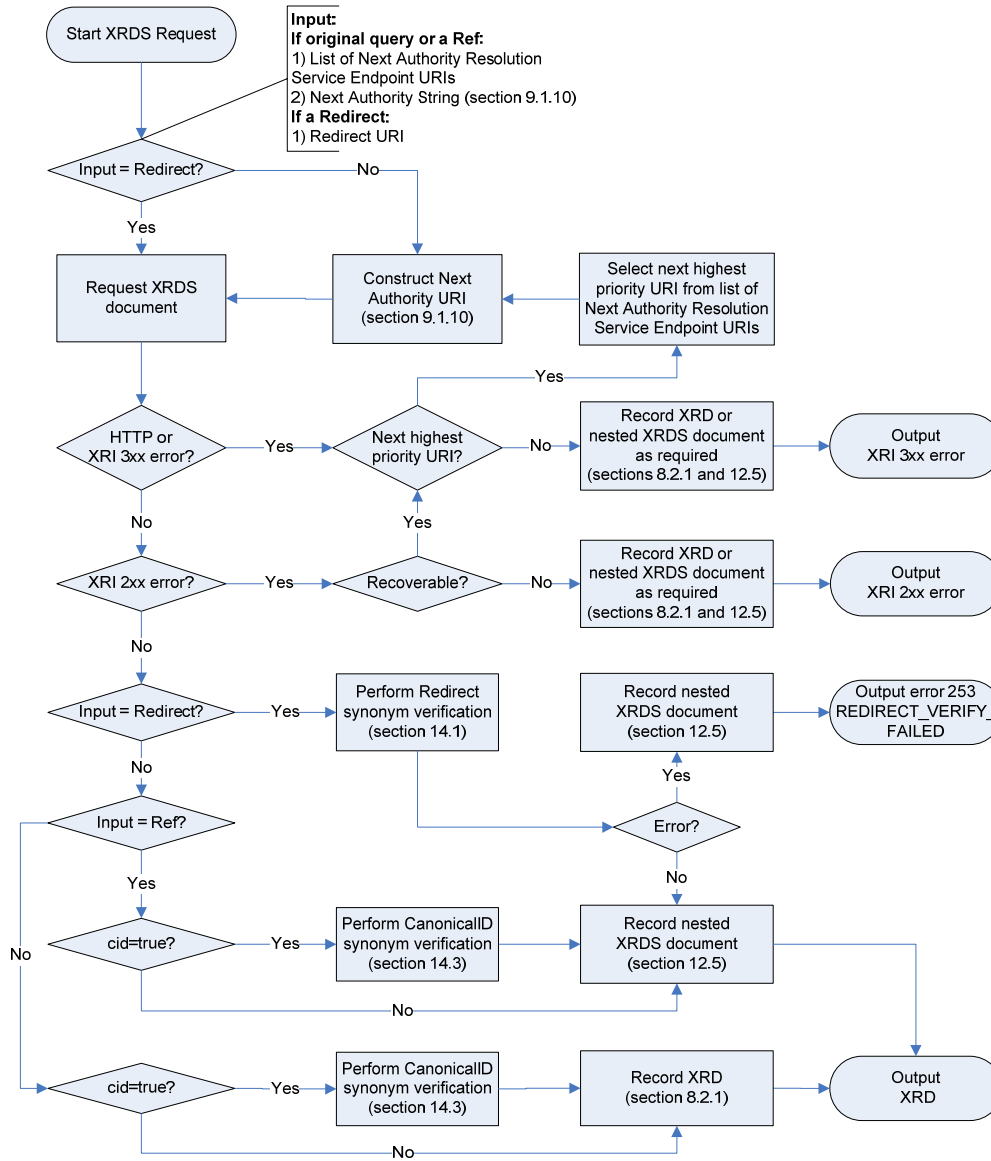
1283

9.1.3 Requesting an XRDS Document using HTTP(S)

1284

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

Comment [DSR7]: This flowchart revised for improved clarity about Redirect URI as an input, synonym verification (for both Redirects, Refs, and standard query XRIs), and recording of XRD and XRDS documents.



1285

1286

1287 Figure 6: XRDS request flowchart.

1288 Following are the normative requirements for an XRI resolver and an XRI authority server when
1289 requesting an XRDS document:

- 1290 1. Each resolution request MUST be an HTTP(S) GET to the Next Authority URI and MUST
1291 contain an Accept header with the media type identifier defined in Table 11. Note that in
1292 XRI authority resolution, this Accept header is NOT interpreted as an XRI resolution input
1293 parameter, but simply as the media type being requested from the server. This differs
1294 from XRI proxy resolution, where the Accept header MAY be used to specify the Service
1295 Media Type resolution parameter. See section 11.5.
- 1296 2. The ultimate HTTP(S) response from an authority server to a successful resolution
1297 request MUST contain either: a) a 2XX response with a valid XRDS document containing
1298 an XRD element for each authority subsegment resolved, or b) a 304 response signifying
1299 that the cached version on the resolver is still valid (depending on the client's HTTP(S)
1300 request). There is no restriction on intermediate redirects (i.e., 3XX result codes) or other
1301 result codes (e.g., a 100 HTTP response) that eventually result in a 2XX or 304 response
1302 through normal operation of [RFC2616].
- 1303 3. The HTTP(S) response from an authority server MUST return the media type requested
1304 by the resolver. The response SHOULD NOT include any subparameters supplied by the
1305 resolver in the request. If the resolver receives such parameters in the response, the
1306 resolver MUST ignore them and do its own independent verification that the response
1307 fulfills the requested parameters.
- 1308 4. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in
1309 the resolution process. In this case, the resolver MUST implement failover handling as
1310 specified in section 9.1.4.
- 1311 5. If all authority resolution service endpoints fail, the resolver SHOULD return the
1312 appropriate error code and context message as specified in section 15. In recursing
1313 resolution, such an error MUST be returned by the recursing authority server to the
1314 resolver as specified in section 15.5.
- 1315 6. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section
1316 16. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent
1317 possible to maintain the efficiency and scalability of the HTTP-based resolution system.
1318 The recommended use of HTTP caching headers is described in more detail in section
1319 16.2.1.

1320 9.1.4 Failover Handling

1321 XRI infrastructure has the same requirements as DNS infrastructure for stability, redundancy, and
1322 network performance. This means XRI authority and proxy resolution services are subject to the
1323 same requirements as DNS nameservers. For example:

- 1324 • Critical authority or proxy resolution servers SHOULD be operated from a minimum of two
1325 physically separate network locations to prevent a single point of failure.
- 1326 • Authority or proxy resolution servers handling heavy loads SHOULD operate from multiple
1327 servers and take advantage of load balancing technologies.

1328 However, such capabilities are effective only if resolvers or other client applications implement
1329 proper failover handling. Because XRI resolution takes place at a layer above DNS resolution,
1330 resolvers have two ways to discover additional network endpoints at which authority or proxy
1331 resolution services are available.

- 1332 • *DNS round robin/failover*: The domain name of an authority resolution service endpoint URI
1333 may be associated with more than one IP address.
- 1334 • *XRI round robin/failover*: The XRDS document describing an XRI authority may publish
1335 multiple URI elements for its authority resolution service endpoint, or multiple authority
1336 resolution service endpoints, or both.

1337 To take advantage of both these options, the following rules apply to failover handling:

- 1338 1. A resolver SHOULD first try an alternate IP address for the current authority resolution
1339 service endpoint if the endpoint uses DNS round robin.
- 1340 2. If all alternate IP addresses fail, a resolver MUST try the next highest priority authority
1341 resolution URI in the current authority resolution service endpoint, if available.
- 1342 3. If all URIs in the current authority resolution service endpoint fail, a resolver MUST try the
1343 next highest priority authority resolution service endpoint, if available, until all authority
1344 resolution service endpoints are exhausted.
- 1345 4. A resolver SHOULD only return an error if all network endpoints associated with the
1346 authority resolution service fail to respond.

1347 **IMPORTANT:** These rules also apply to any client of an XRI proxy resolver. Failure to observe
1348 this warning means the proxy resolver can become a point of failure.

1349 One final consideration: DNS caching mechanisms should respect the TTL (Time To Live)
1350 settings in DNS records. However, different software languages and frameworks handle DNS
1351 caching differently. It is RECOMMENDED to check the default settings to ensure that a library or
1352 application is not caching DNS results indefinitely.

1353 9.1.5 Community Root Authorities

1354 Identifier management policies are defined on a community-by-community basis. For XRI
1355 identifier authorities, the resolution community is specified by the first (leftmost) subsegment of
1356 the authority component of the XRI. This is referred to as the *community root authority*, and it
1357 represents the authority server(s) that answer resolution queries at this root. When a resolution
1358 community chooses to create a new community root authority, it SHOULD define policies for
1359 assigning and managing identifiers under this authority. Furthermore, it SHOULD define what
1360 resolution protocol(s) may be used for these identifiers.

1361 For an XRI authority, the community root may be either a global context symbol (GCS) character
1362 or top-level cross-reference as specified in section 2.2.1.1 of **[XRISyntax]**. In either case, the
1363 corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution
1364 service endpoints for that community.

1365 The community root authority SHOULD publish a self-describing XRDS document as defined in
1366 section 9.1.6. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as
1367 the community's root authority resolution service endpoints. This community root XRDS
1368 document, or its location, must be known *a priori* and is part of the configuration of an XRI
1369 resolver, similar to the specification of root DNS servers for a DNS resolver. Note that it is not
1370 strictly necessary to publish this information in an XRDS document—it may be supplied in any
1371 format that enables configuration of the XRI resolvers in the community. However, publishing a
1372 self-describing XRDS document at a known location simplifies this process and enables dynamic
1373 configuration of community resolvers.

1374 As a best practice, it is RECOMMENDED that community root XRDS document contain:

- 1375 • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- 1376 • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML
1377 trusted resolution is supported.
- 1378 • Both of the above if HTTPS+SAML trusted resolution is supported.
- 1379 • The service endpoints and supported media types of the community's XRI proxy resolver(s) if
1380 proxy resolution is supported.

1381 For a list of public community root authorities and the locations of their community root XRDS
1382 documents, see the Wikipedia entry on XRI **[WikipediaXRI]**.

1383 9.1.6 Self-Describing XRDS Documents

1384 An identifier authority MAY publish a self-describing XRDS document, i.e., one produced by the
1385 same identifier authority that it describes. A resolver MAY request a self-describing XRDS
1386 document from a target identifier authority using either of two methods:

- 1387 1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution
1388 service endpoint, it may use the resolution protocol specified in section 6 to request an
1389 XRDS document directly from this HTTP(S) URI. This HTTP(S) URI may be known a
1390 priori (as is often the case with community root authorities, above), or it may be
1391 discovered from other identifier authorities via the resolution protocols defined in this
1392 specification.
- 1393 2. If the resolver knows: a) an XRI of the target authority as a community root authority, and
1394 b) an HTTP(S) URI for a proxy resolver configured for this community root authority, it
1395 may use the proxy resolution protocol specified in section 11 to query the proxy resolver
1396 for the community root authority XRI. This query MUST include only a single subsegment
1397 identifying the community root authority and MUST NOT include any additional
1398 subsegments.

1399 If an identifier authority had an authority resolution service endpoint at
1400 `http://example.com/auth-res-service/`, an example of the first method would be to
1401 issue an HTTP(S) GET request to that URI with an Accept header specifying the content type
1402 `application/xrds+xml`. See section 6.3 for more details.

1403 If an identifier authority with the community root authority identifier `xri://(example)` was
1404 registered with the XRI proxy resolver `http://xri.example.com/`, an example of the second
1405 method would be to issue an HTTP(S) GET request to the following URI:

1406 `http://xri.example.com/(example)?_xrd_r=application/xrds+xml`

1407 Note that a proxy resolver may use the first method to publish its own self-describing XRDS
1408 document at the HTTP(S) URI(s) for its proxy resolution service.

1409 **IMPORTANT:** A self-describing XRDS document MUST only be issued by an identifier authority
1410 when describing itself. It MUST NOT be included in an XRDS document when describing a
1411 different identifier authority. In the latter case the self-describing XRDS document for the
1412 community root authority is implicit.

1413 9.1.7 Qualified Subsegments

1414 A qualified subsegment is defined by the productions whose names start with `xri-subseg` in
1415 section 2.2.3 of **[XRISyntax]** including the leading syntactic delimiter (“*” or “!”). A qualified
1416 subsegment MUST include the leading syntactic delimiter even if it was optionally omitted in the
1417 original XRI (see section 2.2.3 of **[XRISyntax]**).

1418 If the first subsegment of an XRI authority is a GCS character and the following subsegment does
1419 not begin with a “*” (indicating a reassignable subsegment) or a “!” (indicating a persistent
1420 subsegment), then a “*” is implied and MUST be added when constructing the qualified
1421 subsegment as specified in section 9.1.7. Table 12 and Table 13 illustrate the differences
1422 between parsing a reassignable subsegment following a GCS character and parsing a cross-
1423 reference, respectively.

1424

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

1425 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

1426 Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.

1427 9.1.8 Cross-References

1428 Any subsegment within an XRI authority component may be a cross-reference (see section 2.2.2
 1429 of [XRISyntax]). Cross-references are resolved identically to any other subsegment because the
 1430 cross-reference is considered opaque, i.e., the value of the cross-reference (including the
 1431 parentheses) is the literal value of the subsegment for the purpose of resolution.

1432 Table 14 provides several examples of resolving cross-references. In these examples,
 1433 subsegment !b resolves to a Next Authority Resolution Service Endpoint URI of
 1434 http://example.com/xri/ and recursing authority resolution is not being requested.
 1435

Example XRI	Next Authority URI after resolving
xri://@!a!b!(@!1!2!3)*e/f	http://example.com/xri/!(@!1!2!3)
xri://@!a!b*(mailto:jd@example.com)*e/f	http://example.com/xri/*(mailto:jd@example.com)
xri://@!a!b*(\$v*2.0)*e/f	http://example.com/xri/*(\$v*2.0)
xri://@!a!b*(c*d)*e/f	http://example.com/xri/*(c*d)
xri://@!a!b*(foo/bar)*e/f	http://example.com/xri/*(foo%2Fbar)

Comment [DSR8]: Typo found by Wil Tan.

Deleted: /

1436 Table 14: Examples of the Next Authority URIs constructed using different types of cross-references.

1437 9.1.9 Selection of the Next Authority Resolution Service Endpoint

1438 For each iteration of authority resolution, the resolver MUST select the next authority resolution
 1439 service endpoint from the current XRD as specified in section 13. For generic authority resolution,
 1440 this selection process MUST use the parameters specified in Table 11. For trusted authority
 1441 resolution, this selection process MUST use the parameters specified in Table 15, Table 16, or
 1442 Table 17. In all cases, an explicit match on the xrd:XRD/xrd:Service/xrd:Type element is
 1443 REQUIRED, so during authority resolution, a resolver MUST set the nodefault parameter to a
 1444 value of nodefault=type in order to override selection of a default service endpoint as
 1445 specified in section 13.3.2.

1446 **9.1.10 Construction of the Next Authority URI**

1447 Once the next authority resolution service endpoint is selected, the resolver MUST construct a
1448 URI for the next HTTP(S) request, called the *Next Authority URI*, by concatenating two strings as
1449 specified in this section.

1450 The first string is called the *Next Authority Resolution Service Endpoint URI*. To construct it, the
1451 resolver MUST:

- 1452 1. Select the highest priority URI of the highest priority authority resolution service endpoint
1453 selected in section 9.1.9.
- 1454 2. Apply the service endpoint URI construction algorithm based the value of the `append`
1455 attribute as defined in section 13.7.
- 1456 3. Append a forward slash ("/") *if the URI does not already end in a forward slash*.

1457 The second string is called the *Next Authority String* and it consists of either:

- 1458 • The next fully qualified subsegment to be resolved (see section 9.1.7), or
- 1459 • In the case of recursing resolution, the next fully qualified subsegment to be resolved plus
1460 any additional subsegments for which recursing resolution is requested (see section 9.1.11).

1461 The final step is to append the Next Authority String directly to the Next Authority Resolution
1462 Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

Deleted: to the path component of

1463 BACKWARDS COMPATIBILITY NOTE: Earlier versions of this specification required the Next
1464 Authority String to be appended to the *path component* of the Next Authority Resolution Service
1465 Endpoint URI. This rule was loosened to give XRI authorities greater control over the structure of
1466 incoming resolution requests—for example, to enable Next Authority Strings to appear as query
1467 parameters. Where possible, it is RECOMMENDED that implementations include support for
1468 older resolvers that append the Next Authority String to the path.

Formatted: Font: Italic

1469 Construction of the Next Authority URI is more formally described in this pseudocode for
1470 resolving a "next-auth-string" via a "next-auth-res-sep-uri":

Comment [DSR9]: Added in Revision 02 per suggestion from John Bradley.

```

1471 if (next-auth-res-sep-uri does not end in "/"):
1472     append "/" to next-auth-res-sep-uri
1473
1474 if (next-auth-string is not preceded with "*" or "!" delimiter):
1475     prepend "*" to next-auth-string
1476
1477 append uri-escape(next-auth-string) to next-auth-res-sep-uri

```

Deleted: ¶

Formatted: Font: Bold

Deleted: path portion of

Deleted: path portion of

Deleted: path of

1478 **9.1.11 Recursing Authority Resolution**

1479 If an authority server offers recursing resolution, an XRI resolver MAY request resolution of
1480 multiple authority subsegments in one transaction. If a resolver makes such a request, the
1481 responding authority server MAY perform the additional recursing resolution steps requested. In
1482 this case the recursing authority server acts as a resolver to the other authority resolution service
1483 endpoints that need to be queried. Alternatively, the recursing authority server may retrieve XRDS
1484 from its local cache until it reaches a subsegment whose XRD is not locally cached, or it may
1485 simply recurse only as far as it is authoritative.

1486 If an authority server performs any recursing resolution, it MUST return an ordered list of
1487 `xrd:XRDS` elements (and nested `xrd:XRDS` elements if Redirects or Refs are followed as
1488 specified in section 12) in an `xrd:XRDS` document for all subsegments resolved as defined in
1489 section 8.2.1.

1490 A recursing authority server MAY resolve fewer subsegments than requested by the resolver. The
1491 recursing authority server is under no obligation to resolve more than the first subsegment (for
1492 which it is, by definition, authoritative).

1493 If the recursing authority server does not resolve the entire set of subsegments requested, the
1494 resolver MUST continue the authority resolution process itself. At any stage, however, the
1495 resolver MAY request recursing resolution of any or all of the remaining authority subsegments.

1496 9.2 IRI Authority Resolution

1497 From the standpoint of generic authority resolution, an IRI authority component represents either
1498 a DNS name or an IP address at which an XRDS document describing the authority may be
1499 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET
1500 request to a URI constructed from the IRI authority component. The resulting XRDS document
1501 can then be consumed in the same manner as one obtained using XRI authority resolution.

1502 While the use of IRI authorities provides backwards compatibility with the large installed base of
1503 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of
1504 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities
1505 are NOT RECOMMENDED for new deployments of XRI identifiers.

1506 This section defines IRI authority resolution as a simple extension to the XRI authority resolution
1507 protocol defined in the preceding section.

1508 9.2.1 Service Type and Media Type

1509 Because IRI authority resolution takes place at a level “below” XRI authority resolution, it cannot
1510 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority
1511 resolution uses the same media type as generic XRI authority resolution.

1512 9.2.2 Protocol

1513 Following are the normative requirements for IRI authority resolution that differ from generic XRI
1514 authority resolution:

1515 1. The Next Authority URI (section 9.1.10) is constructed by extracting the entire IRI
1516 authority component and prepending the string `http://`. See the exception in section
1517 9.2.3.

1518 2. The HTTP GET request MUST include an HTTP Accept header containing only the
1519 following:

```
1520 Accept: application/xrds+xml
```

1521 3. The HTTP GET request MUST have a `Host:` header (as defined in section 14.23 of
1522 [RFC2616]) containing the value of the IRI authority component. For example:

```
1523 Host: example.com
```

1524 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document
1525 containing the XRD describing that authority.

1526 5. The responding server MUST use the value of the `Host:` header to populate the
1527 `xrd:XRD/xrd:Query` element in the resulting XRD.

1528 Note that because IRI authority resolution is required to process the entire IRI authority
1529 component in a single step, recursing authority resolution does not apply.

1530 9.2.3 Optional Use of HTTPS

1531 Section 10 of this specification defines trusted resolution only for XRI authorities. Trusted
1532 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to

1533 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY
1534 use HTTPS as the access protocol for retrieving the authority's XRD. If the resolver is satisfied,
1535 via transport level security mechanisms, that the response is from the expected IRI authority, the
1536 resolver MAY consider this an HTTPS trusted resolution response as defined in section 10.1.

1537 10 Trusted Authority Resolution Service

1538 This section defines three options for performing trusted XRI authority resolution as an extension
1539 of the generic authority resolution protocol defined in section 9.1—one using HTTPS, one using
1540 SAML assertions, and one using both.

1541 10.1 HTTPS

1542 HTTPS authority resolution is a simple extension to generic authority resolution in which all
1543 communication with authority resolution service endpoints is carried out over HTTPS. This
1544 provides transport-level security and server authentication, however it does not provide message-
1545 level security or a means for a responder to provide different responses for different requestors.

1546 10.1.1 Service Type and Service Media Type

1547 The protocol defined in this section is identified by the values in Table 15.

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

1548 *Table 15: Service Type and Service Media Type values for HTTPS trusted authority resolution.*

1549 An HTTPS trusted resolution service endpoint advertised in an XRDS document **MUST** use the
1550 Service Type identifier and Service Media Type identifier (including the `https=true` parameter)
1551 defined in Table 15. In addition, the identifier authority **MUST** use an HTTPS URI as the value of
1552 the `xrd:URI` element(s) for this service endpoint.

1553 10.1.2 Protocol

1554 Following are the normative requirements for HTTPS trusted authority resolution that differ from
1555 generic authority resolution (section 9.1):

- 1556 1. All authority resolution service endpoints **MUST** be selected using the values defined in
1557 Table 15.
- 1558 2. All authority resolution requests, including the starting request to a community root
1559 authority, **MUST** use the HTTPS protocol as defined in **[RFC2818]**. This includes all
1560 intermediate redirects, as well as all authority resolution requests resulting from Redirect
1561 and Ref processing as defined in section 12. A successful HTTPS response **MUST** be
1562 received from each authority in the resolution chain or the output **MUST** be error.
- 1563 3. All authority resolution requests **MUST** contain an HTTPS Accept header with the media
1564 type identifier defined in Table 15 (including the `https=true` subparameter).
- 1565 4. If the resolver finds that an authority in the resolution chain does not support HTTPS at
1566 any of its authority resolution service endpoints, the resolver **MUST** return a 23x error as
1567 defined in section 15.

1568 10.2 SAML

1569 In SAML trusted resolution, the resolver uses the Resolution Output Format subparameter
1570 `saml=true` and the authority server responds with an XRDS document containing an XRD with
1571 an additional element—a digitally signed SAML **[SAML]** assertion that asserts the validity of the
1572 containing XRD. SAML trusted resolution provides message integrity but does not provide
1573 confidentiality. For this reason is is **RECOMMENDED** to combine SAML trusted resolution with

1574 HTTPS trusted resolution as defined in section 10.3. Message confidentiality may also be
1575 achieved with other security protocols used in conjunction with this specification. SAML trusted
1576 resolution also does not provide a means for an authority to provide different responses for
1577 different requestors; client authentication is explicitly out-of-scope for version 2.0 of XRI
1578 resolution.

1579 10.2.1 Service Type and Service Media Type

1580 The protocol defined in this section is identified by the values in Table 16.

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

1581 *Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.*

1582 A SAML trusted resolution service endpoint advertised in an XRDS document MUST use the
1583 Service Type identifier and Service Media Type identifier defined in Table 16 (including the
1584 `saml=true` subparameter). In addition, for transport security the identifier authority SHOULD
1585 offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1586 10.2.2 Protocol

1587 10.2.2.1 Client Requirements

1588 For a resolver, trusted resolution is identical to the generic resolution protocol (section 9.1) with
1589 the addition of the following requirements:

- 1590 1. All authority resolution service endpoints MUST be selected using the values defined in
1591 Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an
1592 authority unless the authority advertises a resolution service endpoint matching these
1593 values.
- 1594 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is
1595 RECOMMENDED for confidentiality.
- 1596 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the
1597 media type identifier defined in Table 16 (including the `saml=true` subparameter). This
1598 is the media type of the requested response.

1599 **IMPORTANT:** Clients willing to accept either generic or trusted responses MAY use a
1600 combination of media type identifiers in the Accept header as described in section 14.1 of
1601 [RFC2616]. Media type identifiers SHOULD be ordered according to the client's preference for
1602 the media type of the response. If a client performing generic authority resolution receives an
1603 XRD containing SAML elements, it MAY choose not to validate the signature or perform any
1604 processing of these elements.

- 1605 4. A resolver MAY request recursing authority resolution of multiple subsegments as
1606 defined in section 10.2.3.
- 1607 5. The resolver MUST individually validate each XRD it receives in the resolution chain
1608 according to the rules defined in section 10.2.4. When `xrd:XRD` elements come both
1609 from freshly-retrieved XRDS documents and from a local cache, a resolver MUST ensure
1610 that these requirements are satisfied each time a resolution request is performed.

1611 10.2.2.2 Server Requirements

1612 For an authority server, trusted resolution is identical to the generic resolution protocol (section
1613 9.1) with the addition of the following requirements:

- 1614 1. The HTTP(S) response to a trusted resolution request MUST include a content type of
1615 `application/xrds+xml;saml=true`.
- 1616 2. The XRDS document returned by the resolution service MUST contain a
1617 `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid
1618 per the processing rules described by [SAML].
- 1619 3. The `saml:Assertion` element MUST contain a valid enveloped digital signature as
1620 defined by [XMLDSig] and as constrained by section 5.4 of [SAML].
- 1621 4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML
1622 assertion. Specifically, the signature MUST contain a single
1623 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference
1624 MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML
1625 assertion. The URI reference MUST NOT be empty and it MUST refer to the identifier
1626 contained in the `xrd:XRD/@xml:id` attribute.
- 1627 5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain
1628 a `ds:KeyInfo` element. If this element is included, it MUST describe the key used to
1629 verify the digital signature element. However, because the signing key is known in
1630 advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the
1631 `ds:Signature` element of the SAML assertion.
- 1632 6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST
1633 match the XRI authority subsegment requested by the client.
- 1634 7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match
1635 the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD
1636 advertising availability of trusted resolution service from this authority as required in
1637 section 10.2.5.
- 1638 8. The `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
1639 present and equal to the `xrd:XRD/xrd:Query` element.
- 1640 9. The `NameQualifier` attribute of the
1641 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be
1642 present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.
- 1643 10. There MUST be exactly one `saml:AttributeStatement` present in the
1644 `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute`
1645 element with a `Name` attribute value of `xri://$xrd*($v*2.0)`. This
1646 `saml:Attribute` element MUST contain exactly one `saml:AttributeValue`
1647 element whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD`
1648 element that is the immediate parent of the `saml:Assertion` element.

1649 10.2.3 Recursing Authority Resolution

1650 If a resolver requests trusted resolution of multiple authority subsegments (see section 9.1.8), a
1651 recursing authority server SHOULD attempt to perform trusted resolution on behalf of the resolver
1652 as described in this section. However, if the resolution service is not able to obtain trusted XRDs
1653 for one or more additional recursing subsegments, it SHOULD return only the trusted XRDs it has
1654 obtained and allow the resolver to continue.

1655 10.2.4 Client Validation of XRDs

1656 For each XRD returned as part of a trusted resolution request, the resolver MUST validate the
1657 XRD according to the rules defined in this section.

- 1658 1. The `xrd:XRD/saml:Assertion` element MUST be present.
- 1659 2. This assertion MUST be valid per the processing rules described by [SAML].
- 1660 3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by
1661 [XMLDSig] and constrained by Section 5.4 of [SAML].
- 1662 4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML
1663 assertion. Specifically, the signature MUST contain a single
1664 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference
1665 MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent
1666 of the signed SAML assertion.
- 1667 5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo`
1668 element, the resolver MAY reject the signature if this key does not match the signer's
1669 expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor
1670 that was used to describe the current authority. See section 10.2.5.
- 1671 6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose
1672 resolution resulted in the current XRD.
- 1673 7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1674 `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability
1675 of trusted resolution service from this authority as required in section 10.2.5.
- 1676 8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the
1677 `NameQualifier` attribute of the
1678 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1679 9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the
1680 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1681 10. There MUST exist exactly one
1682 `xrd:XRD/saml:Assertion/saml:AttributeStatment` with exactly one
1683 `saml:Attribute` element that has a `Name` attribute value of `xri://$xrd*($v*2.0)`.
1684 This `saml:Attribute` element must have exactly one `saml:AttributeValue`
1685 element whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD`
1686 element that is the immediate parent of the signed SAML assertion.

1687 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result
1688 MUST NOT be considered a valid trusted resolution response as defined by this specification.
1689 Note that this does not preclude a resolver from considering alternative resolution paths. For
1690 example, if an XRD advertising SAML trusted resolution service has two or more
1691 `xrd:XRD/xrd:Service/xrd:URI` elements and the response from one service endpoint fails
1692 to meet the requirements above, the client MAY repeat the validation process using the second
1693 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as
1694 defined by this document and SAML trusted resolution may continue.

1695 If the above requirements are met, and the `code` attribute of the `xrd:XRD/xrd:ServerStatus`
1696 element is 100 (SUCCESS), the resolver MUST add an `xrd:XRD/xrd:Status` element
1697 reporting a status of 100 (SUCCESS) as specified in section 15. Note that this added element
1698 MUST be disregarded if a consuming application wishes to verify the SAML signature itself. (If
1699 necessary, the consuming application may request the XRDS document it wishes to verify directly
1700 from the SAML authority resolution server.)

1701 If all SAML trusted resolution paths fail, the resolver MUST return the appropriate 23x trusted
1702 resolution error as defined in section 15.

1703 10.2.5 Correlation of ProviderID and KeyInfo Elements

1704 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at
1705 least one unique persistent identifier expressed in the

1706 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority
1707 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI
1708 authority. While a ProviderID may be any valid URI that meets these requirements, it is
1709 STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI
1710 **[XRISyntax]** or a URN **[RFC2141]**.

1711 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in
1712 an XRD advertising SAML trusted authority resolution service with the response received from a
1713 SAML trusted resolution service endpoint. If the signed XRD response contains the same
1714 ProviderID as the XRD used to advertise a service, and the resolver has reason to trust the
1715 signature, the resolver can trust that the XRD response has not been maliciously replaced with
1716 another XRD.

1717 There is no defined discovery process for the ProviderID for a community root authority; it must
1718 be published in a self-describing XRDS document (or other equivalent description—see sections
1719 9.1.5 and 9.1.6) and verified independently. Once the community root XRDS document is known,
1720 the ProviderID for delegated XRI authorities within this community MAY be discovered using the
1721 `xrd:XRD/xrd:Service/xrd:ProviderID` element of authority resolution service endpoints.
1722 This trust mechanism MAY also be used for other services offered by an authority.

1723 In addition, the metadata necessary for SAML trusted authority resolution or other SAML **[SAML]**
1724 interactions MAY be discovered using the `ds:KeyInfo` element (section 4.2.) Again, if this
1725 element is present in an XRD advertising SAML authority resolution service (or any other
1726 service), and the client has reason to trust this XRD, the client MAY use the associated
1727 ProviderID to correlate the contents of this element with a signed response.

1728 To assist resolvers in using this key discovery mechanism, it is important that trusted authority
1729 servers be configured to sign responses in such a way that the signature can be verified using the
1730 correlated `ds:KeyInfo` element. For more information, see **[SAML]**.

1731 10.3 HTTPS+SAML

1732 10.3.1 Service Type and Service Media Type

1733 The protocol defined in this section is identified by the values in Table 17.

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

1734 *Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.*

1735 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST
1736 use the Service Type identifier and Service Media Type identifier defined in Table 17 (including
1737 the `https=true` and `saml=true` subparameters). In addition, the identifier authority MUST use
1738 an HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1739 **10.3.2 Protocol**

1740 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- 1741 1. All authority resolution service endpoints MUST be selected using the values defined in
1742 Table 17.
- 1743 2. All authority resolution requests and responses, including the starting request to a
1744 community root authority, MUST conform to both the requirements of the HTTPS trusted
1745 resolution protocol defined in section 10.1 and the SAML trusted resolution protocol
1746 defined in section 10.2.
- 1747 3. All authority resolution requests MUST contain an HTTPS Accept header with the media
1748 type identifier defined in Table 17 (including both the `https=true` and `saml=true`
1749 parameters). This MUST be interpreted as the value of the Resolution Output Format
1750 input parameter.
- 1751 4. If the resolver finds that an authority in the resolution chain does not support both HTTPS
1752 and SAML, the resolver MUST return a 23x error as defined in section 15.

11 Proxy Resolution Service

The preceding sections have defined XRI resolution as a set of logical functions. 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 11.2. HXRIs also enable XRI resolution input parameters to be encoded as query parameters in the HXRI.

Proxy resolution is useful for:

- Offloading XRI resolution and service endpoint selection processing from a client to an HTTP(S) server.
- 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 16.4.
- Returning HTTP(S) redirects to clients such as browsers that have no native understanding of XRIs but can process HXRIs. This provides backwards compatibility with the large installed base of existing HTTP clients.

11.1 Service Type and Media Types

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

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

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

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. In addition:

- An HTTPS proxy resolver MUST specify the media type parameter `https=true` and MUST offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.
- A SAML proxy resolver MUST specify the media type parameter `saml=true` and SHOULD offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

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 9.1.3 and 9.1.6) 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 resolution capabilities of a proxy resolver from this source.

11.2 HXRIs

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:

- 1790 • It allows XRIs to be used anywhere an HTTP URI can appear, including in Web pages,
1791 electronic documents, email messages, instant messages, etc.
- 1792 • It allows XRI-aware processors and search agents to recognize an HXRI and extract the
1793 embedded XRI for direct resolution, processing, and indexing.

1794 To make this syntax as simple as possible for XRI-aware processors or search agents to
1795 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority component that
1796 begins with the domain name segment "xri.". The QXRI is then appended as the entire local
1797 path (and query component, if present). The QXRI MUST NOT include the xri:// prefix and
1798 MUST be in URI-normal form as defined in [XRISyntax]. (If a proxy resolver receives an HXRI
1799 containing a QXRI beginning with an xri:// prefix, it SHOULD remove it before continuing.) In
1800 essence, the proxy resolver URI (including the forward slash after the domain name) serves as a
1801 machine-readable alternate prefix for an absolute XRI in URI-normal form.

1802 The normative ABNF for an HXRI is defined below based on the ireg-name, xri-hier-part,
1803 and iquery productions defined in [XRISyntax]. XRIs that need to be understood by non-XRI-
1804 aware clients SHOULD be published as HTTP URIs conforming to this HXRI production.

```
1805 HXRI           = proxy-resolver "/" QXRI
1806 proxy-resolver = ( "http://" / "https://" ) proxy-reg-name
1807 proxy-reg-name = "xri." ireg-name
1808 QXRI           = xri-hier-part [ "?" i-query ]
```

1809 URI processors that recognize XRIs SHOULD interpret the local part of an HTTP or HTTPS URI
1810 (the path segment(s) and optional query segment) as an XRI provided that: a) it conforms to this
1811 ABNF, and b) the first segment of the path conforms to the xri-authority or iauthority productions
1812 in [XRISyntax].

1813 For references to communities that offer public XRI proxy resolution services, see the Wikipedia
1814 entry on XRI [WikipediaXRI].

1815 11.3 HXRI Query Parameters

1816 In proxy resolution, the XRI resolution input parameters defined in section 8.1 are bound to an
1817 HTTP(S) interface using the conventional web model of encoding them in an HTTP(S) URI, which
1818 in this case is an HXRI. The binding of the logical parameter names to HXRI component parts is
1819 defined in Table 19.

Logical Parameter Name	HXRI Component	HXRI Query Parameter Name
QXRI	Entire path and query string of HXRI (exclusive of HXRI query parameters listed below)	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

1820 Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.

- 1821 Following are the rules for the use of the parameters specified in Table 19.
- 1822 1. The QXRI MUST be normalized as specified in section 11.2.
- 1823 2. If the original QXRI has an existing query component, the HXRI query parameters MUST
- 1824 be appended to that query component.
- 1825 **IMPORTANT:** The query parameter names in Table 19 were chosen to minimize the probability of
- 1826 collision with any pre-existing query parameter names in the QXRI. If there is any conflict, the
- 1827 pre-existing query parameter names MUST be percent-encoded prior to transformation into an
- 1828 HXRI.
- 1829 3. After proxy resolution, the HXRI query parameters MUST subsequently be removed from
- 1830 the QXRI query component. The existing QXRI query component MUST NOT be altered
- 1831 in any other way, i.e., it must be passed through with no changes in parameter order,
- 1832 escape encoding, etc.
- 1833 4. If the original QXRI does not have a query component, one MUST be added to pass any
- 1834 HXRI query parameters. After proxy resolution, this query component MUST be entirely
- 1835 removed.
- 1836 5. If the original QXRI had a null query component (only a leading question mark), or a
- 1837 query component consisting of only question marks, *one additional leading question mark*
- 1838 MUST be added before adding any HXRI query parameters. After proxy resolution, any
- 1839 HXRI query parameters and exactly one leading question mark MUST be removed. See
- 1840 the URI construction steps defined in section 13.6.
- 1841 6. Each HXRI query parameter MUST be delimited from other parameters by an ampersand
- 1842 (“&”).
- 1843 7. Each HXRI query parameter MUST be delimited from its value by an equals sign (“=”).
- 1844 8. If an HXRI query parameter includes one of the media type parameters defined in Table
- 1845 6, it MUST be delimited from the HXRI query parameter with a semicolon (“;”).
- 1846 9. The fully-composed HXRI MUST be encoded and decoded as specified in section 11.4.
- 1847 10. If any HXRI query parameter name is included but its value is empty, the value of the
- 1848 parameter MUST be considered null.

1849 **11.4 HXRI Encoding/Decoding Rules**

1850 To conform with the typical requirements of web server URI parsing libraries, HXRIs MUST be

1851 encoded prior to input to a proxy resolver and decoded prior to output from a proxy resolver.

1852 Because web server libraries typically perform some of these decoding functions automatically,

1853 implementers MUST ensure that a proxy resolver, when used in conjunction with a specific web

1854 server, accomplishes the full set of HXRI decoding steps specified in this section. In particular,

1855 these decoding steps MUST be performed prior to any comparison operations defined in this

1856 specification.

1857 Before any HXRI-specific encoding steps are performed, the QXRI portion of the HXRI (including

1858 all HXRI query parameters) MUST be transformed into URI-normal form as defined in section 2.3

1859 of **[XRISyntax]**. This means characters not allowed in URIs, such as SPACE, or characters that

1860 are valid only in IRIs, such as UCS characters above the ASCII range, MUST be percent

1861 encoded. Also, the plus sign character (“+”) MUST NOT be used to encode the SPACE character

1862 because in decoding the percent-encoded sequence %2B MUST be interpreted as the plus sign

1863 character (“+”).

1864 Once the HXRI is in URI-normal form, the following sequence of encoding steps MUST be

1865 performed in the order specified before an HXRI is submitted to a proxy resolver.

1866 **IMPORTANT:** this sequence of steps is not idempotent, so it MUST be performed only once.

- 1867 1. First, in order to preserve percent-encoding when the HXRI is passed through a web
 1868 server, all percent signs MUST be themselves percent-encoded, i.e., a SPACE encoded
 1869 as %20 will become %2520.
- 1870 2. Second, to prevent misinterpretation of HXRI query parameters, any occurrences of the
 1871 ampersand character (“&”) within an HXRI query parameter that are NOT used to delimit
 1872 it from another query parameter MUST be percent encoded using the sequence %26.
- 1873 3. Third, to prevent misinterpretation of the semicolon character by the web server, any
 1874 semicolon used to delimit one of the media type parameters defined in Table 6 from the
 1875 media type value MUST be percent-encoded using the sequence %3B.

1876 To decode an encoded HXRI back into URI-normal form, the above sequence of steps MUST be
 1877 performed in reverse order. Again, the sequence is not idempotent so it MUST be performed only
 1878 once.

1879 Table 20 illustrates the components of an example HXRI before transformation to URI-normal
 1880 form. The characters requiring percent encoding are highlighted in **red**. Note the space in the
 1881 string `hello planète`. Also, for purposes of illustration, the Type component contains a query
 1882 string (which would not normally appear in a Type identifier).

QXRI	<code>https://xri.example.com/=example*r^Esum^E/path?query</code>
<code>_xrd_r</code>	<code>_xrd_r=application/xrds+xml;https=true;sep=true</code>
<code>_xrd_t</code>	<code>_xrd_t=http://example.org/test?a=1&b=hello planète</code>
<code>_xrd_m</code>	<code>_xrd_m=application/atom+xml</code>

1883 *Table 20: Example of HXRI components prior to transformation to URI-normal form.*

1884 Table 21 illustrates these components after transformation to URI-normal form. Characters that
 1885 have been percent-encoded are in **blue**. Characters still requiring percent encoding according to
 1886 the rules defined in this section are highlighted in **red**.

QXRI	<code>https://xri.example.com/=example*r^{E9}sum^{E9}/path?query</code>
<code>_xrd_r</code>	<code>_xrd_r=application/xrds+xml;https=true;sep=true</code>
<code>_xrd_t</code>	<code>_xrd_t=http://example.org/test?a=1&b=hello%20plan^{E8}te</code>
<code>_xrd_m</code>	<code>_xrd_m=application/atom+xml</code>

1887 *Table 21: Example of HXRI components after transformation to URI-normal form.*

1888 Table 22 illustrates the components after all encoding rules defined in this section are applied.

QXRI	<code>https://xri.example.com/=example*r^{25E9}sum^{25E9}/path?query</code>
<code>_xrd_r</code>	<code>_xrd_r=application/xrds+xml^{3B}https=true^{3B}sep=true</code>
<code>_xrd_t</code>	<code>_xrd_t=http://example.org/test?a=1²⁶b=hello%2520plan^{25E8}te</code>
<code>_xrd_m</code>	<code>_xrd_m=application/atom+xml</code>

1889 *Table 22: Example of HXRI components after application of the required encoding rules.*

1890 Following is the fully-encoded HXRI:

```
1891 https://xri.example.com/=example*r%25E9sum%25E9/path?query
1892 &_xrd_r=application/xrds+xml%3Bhttps=true%3Bsep=true
1893 &_xrd_t=http://example.org/test?a=1%26b=hello%2520plan%25E8te
1894 &_xrd_m=application/atom+xml
```

1895 Following is the fully decoded HXRI returned to URI-normal form. Note that the proxy resolver
1896 MUST leave the HXRI in URI-normal form for any further processing.

```
1897 https://xri.example.com/=example*r%E9sum%E9/path?query
1898 &_xrd_r=application/xrds+xml;https=true;sep=true
1899 &_xrd_t=http://example.org/test?a=1&b=hello%20plan%E8te
1900 &_xrd_m=application/atom+xml
```

1901 11.5 HTTP(S) Accept Headers

1902 In proxy resolution, one XRI resolution input parameter, the Service Media Type (section 8.1.4)
1903 MAY be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. The
1904 following rules apply to this input:

- 1905 1. As described in section 14.1 of **[RFC2616]**, the Accept header content type MAY consist
1906 of multiple media type identifiers. If so, the proxy resolver MUST choose only one to
1907 accept. A proxy resolver client SHOULD order media type identifiers according to the
1908 client's preference and a proxy resolver server SHOULD choose the client's highest
1909 preference.
- 1910 2. If the value of the Accept header content type is null, this MUST be interpreted as the
1911 value of the Service Media Type parameter.
- 1912 3. If the value of the Service Media Type parameter is explicitly set via the `_xrd_m` query
1913 parameter in the HXRI (including to a null value), this MUST take precedence over any
1914 value set via an HTTP(S) Accept header.

1915 11.6 Null Resolution Output Format

1916 Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a
1917 resolution request where the Resolution Output Format input parameter value is null—either
1918 because this parameter is absent or because it was explicitly set to null using the `_xrd_r` query
1919 parameter.

1920 If the value of the Resolution Output Format value is null, a resolver MUST proceed as if the
1921 following media type parameters had the following values: `https=false`, `saml=false`,
1922 `refs=true`, `sep=true`, `nodefault_t=false`, `nodefault_p=false`,
1923 `nodefault_m=false`, and `uric=false`. In addition, the output MUST be an HTTP(S) redirect
1924 as defined in the following section.

1925 11.7 Outputs and HTTP(S) Redirects

1926 For all values of the Resolution Output Format parameter except null, a proxy resolver MUST
1927 follow the output rules defined in section 8.2.

1928 If the value of the Resolution Output Format is null, and the output is not an error, a proxy
1929 resolver MUST follow the rules for output of a URI List as defined in section 8.2.3. However,
1930 instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as
1931 an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service
1932 Media Type parameter.

1933 If the output is an error, a proxy resolver SHOULD return a human-readable error message as
1934 specified in section 15.4.

1935 These rules enable XRI proxy resolvers to serve clients that do not understand XRI syntax or
1936 resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the
1937 service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept
1938 header (if any).

1939 **11.8 Differences Between Proxy Resolution Servers**

1940 An XRI proxy resolution request MAY be sent to any proxy resolver that will accept it. All XRI
1941 proxy resolvers SHOULD deliver uniform responses given the same QXRI and other input
1942 parameters. However, because proxy resolvers may potentially need to make decisions about
1943 network errors, Redirect and Ref processing, and trust policies on behalf of the client they are
1944 proxying, and these decisions may be based on local policy, in some cases different proxy
1945 resolvers may return different results.

1946 **11.9 Combining Authority and Proxy Resolution Servers**

1947 The majority of DNS nameservers are recursing nameservers that answer both queries for which
1948 they are authoritative and queries which they must forward to other nameservers. The same rule
1949 applies in XRI architecture: in many cases the optimum configuration will be combining an
1950 authority server and proxy resolver in the same server. This server can publish a self-describing
1951 XRDS document (section 9.1.6) that advertises both its authority resolution and proxy resolution
1952 service endpoints. It can also optimize caching of XRDs for clients in its resolution community
1953 (see section 16.4).

1954

12 Redirect and Ref Processing

1955
1956
1957

The purpose of the `xrd:Redirect` and `xrd:Ref` elements is to enable identifier authorities to distribute and delegate management of XRDS documents. There are two primary use cases for using multiple XRDS documents to describe the same resource:

1958
1959
1960

- One identifier authority needs to manage descriptions of the resource from different physical locations on the network, e.g., registry, directory, webserver, blog, etc. This is the purpose of the `xrd:Redirect` element.

1961
1962
1963
1964

- One identifier authority needs to delegate all or part of resource description to a different identifier authority, e.g., an individual might delegate responsibility for different aspects of an XRDS to his/her spouse, school, employer, doctor, etc. This is the purpose of the `xrd:Ref` element.

1965
1966

Table 23 summarizes the similarities and differences between the `xrd:Redirect` and `xrd:Ref` 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 identifier authority	No	Yes
Must include a subset of the synonyms available in the source XRD	Yes	No
Available at both XRD level and SEP level	Yes	Yes
Processed automatically if present at the XRD level	Yes	Yes
Always results in nested XRDS document, even if only to report an error	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

1967

Table 23: Comparison of Redirect and Ref elements.

1968
1969

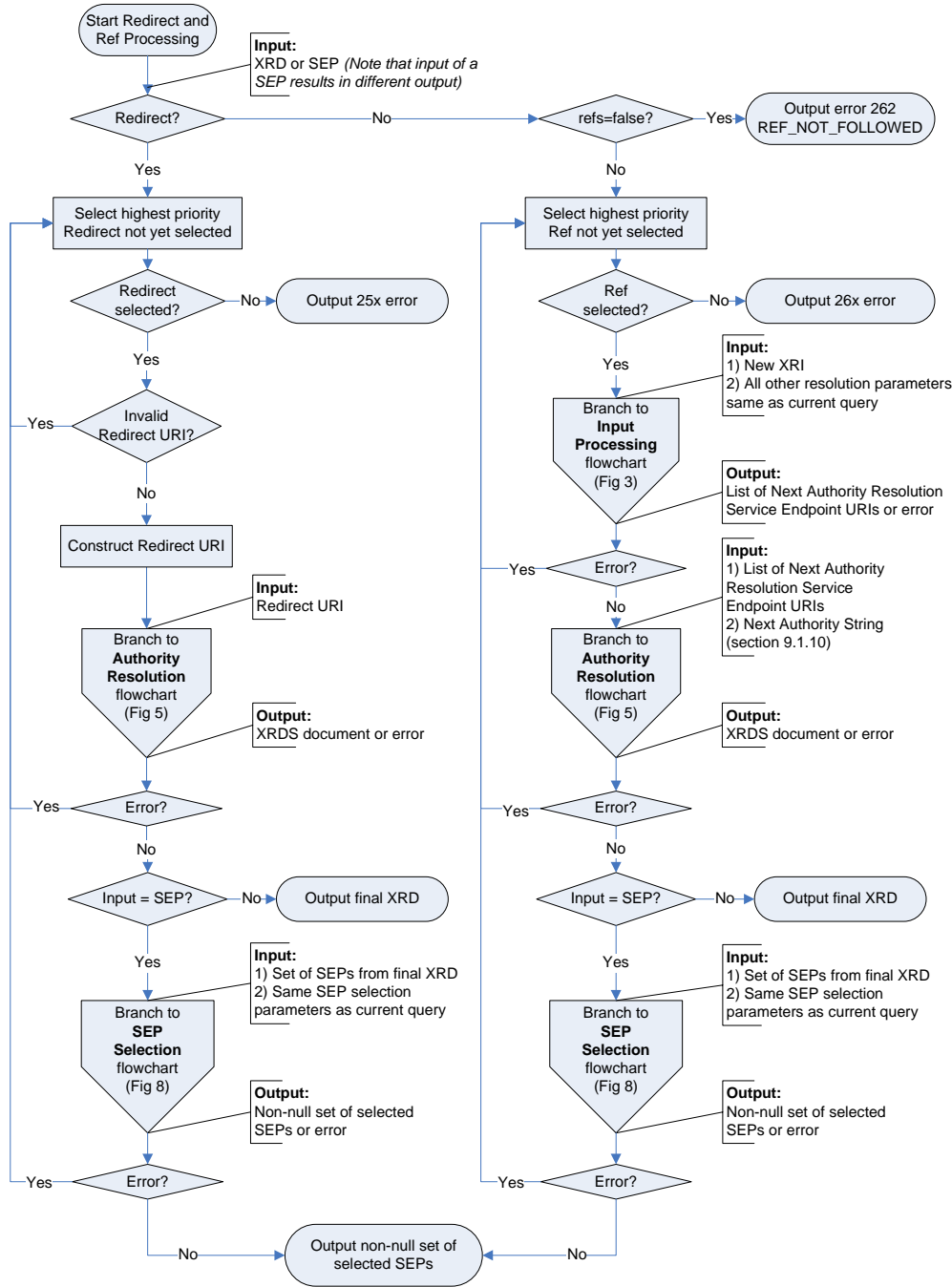
The combination of Redirect and Ref elements should enable identifier authorities to implement a wide variety of distributed XRDS management policies.

1970
1971
1972
1973
1974

IMPORTANT: Since they involve recursive calls, XRDS authors SHOULD use Redirects and Refs carefully and SHOULD perform special testing on XRDS documents containing Redirects and/or Refs to ensure they yield expected results. In particular implementers should study the recursive calls between authority resolution and service endpoint selection illustrated in Figure 2, Figure 5, Figure 7, and Figure 8 and see the guidance in section 12.6, *Recursion and Backtracking*.

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

Comment [DSR10]: This flowchart revised for improved clarity about Redirect URI as an input, outputs, and recording of XRD and XRDS documents.



1976
1977

1978 Figure 7: Redirect and Ref processing flowchart.

1979 12.1 Cardinality

1980 Redirect and Ref elements may be used both at the XRD level (as a child of the `xrd:XRD`
1981 element) and the SEP level (as a child of the `xrd:XRD/xrd:Service` element) within an XRD.
1982 In both cases, to simplify processing, the XRD schema (Appendix B) enforces the following rules:

- 1983 • At the XRD level, an XRD MAY contain only one of the following: zero-or-more
1984 `xrd:Redirect` or zero-or-more `xrd:Ref` elements.
- 1985 • At the SEP level, a SEP MAY contain only one of the following: zero-or-more `xrd:URI`
1986 elements, zero-or-more `xrd:Redirect` elements, or zero-or-more `xrd:Ref` elements.

Comment [DSR11]: Clarified wording per comment from Eran Hammer-Lahav.

1987 12.2 Precedence

1988 XRDS authors should take special note of the following precedence rules for Redirect and Refs.

- 1989 1. If a Redirect or Ref element is present at the XRD level, it MUST be processed
1990 immediately before a resolver continues with authority resolution, performs service
1991 endpoint selection (required or optional), or returns its final output. This rule applies
1992 recursively to all XRDS documents resolved as a result of Redirect or Ref processing.
- 1993 2. If a Redirect or Ref element is not present at the XRD level, but is present in the highest
1994 priority service endpoint selected by the rules in section 13, it MUST be processed
1995 immediately before a resolver completes service endpoint selection (required or optional),
1996 or returns its final output. This rule also applies recursively to all XRDS documents
1997 resolved as a result of Redirect or Ref processing.

1998 **IMPORTANT:** Due to these rules, even if a resolver has resolved the final subsegment of an XRI,
1999 the authority resolution phase is still not complete as long as the final XRD has a Redirect or Ref
2000 at the XRD level. This Redirect or Ref MUST be resolved until it returns an XRD that does not
2001 contain an Redirect or Ref at the XRD level. The same rule applies to the optional service
2002 endpoint selection phase: it is not complete until it locates a final XRD that contains the requested
2003 SEP but: a) the XRD does not contain an Redirect or Ref at the XRD level, and b) the highest
2004 priority selected SEP does not contain a Redirect or Ref.

2005 Based on these rules, the following best practices are recommended.

- 2006 1. XRDS authors SHOULD NOT put any service endpoints in an XRD that contains a
2007 Redirect or Ref at the XRD level because by definition these service endpoints will be
2008 ignored.
- 2009 2. XRDS authors SHOULD use a Redirect or Ref element at the XRD level if they wish to
2010 relocate or delegate resolution behavior regardless of any service endpoint query.
- 2011 3. XRDS authors SHOULD use a Redirect or Ref element in a service endpoint for which
2012 they expect a POSITIVE match as defined in section 13.4.1 if they wish to control
2013 resolution behavior based an explicit service endpoint match.
- 2014 4. XRDS authors SHOULD use a Redirect or Ref element in a service endpoint for which
2015 they expect a DEFAULT match as defined in section 13.4.1 if they wish to control
2016 resolution behavior based on the absence of an explicit service endpoint match.
- 2017 5. XRDS authors SHOULD NOT include two or more SEPs of equal priority in an XRD if
2018 they contain Redirects or Refs that will make resolution ambiguous or non-deterministic.

2019 Also note that, during the authority resolution phase, a Redirect or Ref placed in the highest
2020 priority authority resolution SEP of an XRD will have effectively the same result as a Redirect or
2021 Ref placed at the XRD level. The first option (placement in the SEP) SHOULD be used if the XRD
2022 contains other service endpoints or metadata describing the resource. The second option
2023 (placement at the XRD level) SHOULD be used only if the XRD contains no service endpoints.

2024 12.3 Redirect Processing

2025 The purpose of the `xrd:Redirect` element is to enable an authority to redirect from an XRDS
2026 document managed in one network location (e.g., a registry) to a different XRDS document
2027 managed in a different network location by the same authority (e.g., a web server, blog, etc.) It is
2028 similar to an HTTP(S) redirect; however, it is managed at the XRDS document level rather than
2029 HTTP(S) transport level. Note that unlike a Ref, a Redirect does NOT delegate to a different XRI
2030 authority, but only to the same authority at a different network location.

2031 Following are the normative rules for processing of the `xrd:Redirect` element.

- 2032 1. To process a Redirect at either the XRD or SEP level, the resolver MUST begin by
2033 selecting the highest priority `xrd:XRD/xrd:Redirect` element in the XRD or SEP.
- 2034 2. If the value of the resolution subparameter `https` is FALSE, or the subparameter is
2035 absent or empty, the value of the selected `xrd:Redirect` element MUST be EITHER a
2036 valid HTTP URI or a valid HTTPS URI. If not, the resolver MUST select the next highest
2037 priority `xrd:Redirect` element. If all instances of this element fail, the resolver MUST
2038 stop and return the error 251 `INVALID_REDIRECT` in the XRD containing the Redirect
2039 or as a plain text error message as specified in section 15.
- 2040 3. If the value of the resolution subparameter `https` is TRUE, the value of the selected
2041 `xrd:Redirect` element MUST be a valid HTTPS URI. If not, the resolver MUST select
2042 the next highest priority `xrd:Redirect` element. If all instances of this element fail, the
2043 resolver MUST stop and return the error 252 `INVALID_HTTPS_REDIRECT` in the XRD
2044 containing the Redirect or as a plain text error message as specified in section 15.
- 2045 4. Once a valid `xrd:Redirect` element has been selected, if the
2046 `xrd:XRD/xrd:Redirect` element includes the `append` attribute, the resolver MUST
2047 construct the final HTTP(S) URI as defined in section 13.7.
- 2048 5. The resolver MUST request a new XRDS document from the final HTTP(S) URI using the
2049 protocol defined in section 9.1.3. If the Resolution Output Format is an XRDS document,
2050 the resolver MUST embed a nested XRDS document containing an XRD representing
2051 the Redirect as specified in section 12.5.
- 2052 6. If resolution of an `xrd:Redirect` element fails during the authority resolution phase of
2053 the original resolution query, or if resolution of an `xrd:Redirect` element fails during
2054 the optional service endpoint selection phase OR the final XRD does not contain the
2055 requested SEP, then the resolver MUST report the error in the final XRD of the nested
2056 XRDS document using the status codes defined in section 15. (One nested XRDS
2057 document will be added for each Redirect attempted by the resolver.) The resolver MUST
2058 then select the next highest priority `xrd:Redirect` element from the original XRD or
2059 SEP and repeat rule 7. For more details, see section 12.6, *Recursion and Backtracking*.
- 2060 7. If resolution of all `xrd:Redirect` elements in the XRD or SEP that originally triggered
2061 Redirect processing fails, the resolver MUST stop and return a 25x error in the XRD
2062 containing the Redirect or as a plain text error message as specified in section 15. The
2063 resolver MUST NOT try any other SEPs even if multiple SEPs were selected as specified
2064 in section 13.
- 2065 8. If resolution succeeds, the resolver MUST verify the synonym elements in the new XRD
2066 as specified in section 14.1. If synonym verification fails, the resolver MUST stop and
2067 return the error specified in that section.
- 2068 9. If the value of the resolution subparameter `saml` is TRUE, the resolver MUST verify the
2069 signature on the XRD as specified in section 10.2.4. If signature verification fails, the
2070 resolver MUST stop and return the error specified in that section.
- 2071 10. If Redirect resolution succeeds, further authority resolution or service endpoint selection
2072 MUST continue based on the new XRD.

Comment [DSR12]: Corrected reference from section 6.3 to 9.1.3 (caught by Wil Tan).

2073 12.4 Ref Processing

2074 The purpose of the `xrd:Redirect` element is to enable one authority to delegate management
2075 of all or part of an XRDS document to another authority. For example, an individual might
2076 delegate management of all or portions of an XRDS document to his/her spouse, school,
2077 employer, doctor, etc. This delegation may cover the entire document (an XRD level Ref), or only
2078 one or more specific service endpoints within the document (a SEP level Ref).

2079 Following are the normative rules for processing of the `xrd:Ref` element.

- 2080 1. Ref processing is **only performed** if the value of the `refs` subparameter (Table 6) is
2081 TRUE or it is absent or empty. If the value is FALSE and the XRD contains at least one
2082 `xrd:Ref` element that could be followed to complete the resolution query, the resolver
2083 MUST stop and return the error 262 `REF_NOT_FOLLOWED` in the XRD containing the
2084 Ref or as a plain text error message as defined in section 15. The rules below presume
2085 that `refs=true`.
- 2086 2. To process a Ref at either the XRD or SEP level, the resolver MUST begin by selecting
2087 the highest priority `xrd:XRD/xrd:Ref` element from the XRD or SEP.
- 2088 3. The value of the selected `xrd:Ref` element MUST be a valid absolute XRI. If not, the
2089 resolver MUST select the next highest priority `xrd:Ref` element. If all instances of this
2090 element fail, the resolver MUST stop and return the error 261 `INVALID_REF` in the XRD
2091 containing the Ref or as a plain text error message as defined in section 15.
- 2092 4. Once a valid `xrd:XRD/xrd:Ref` value is selected, the resolver MUST begin resolution
2093 of a new XRDS document from this XRI using the protocols defined in this specification.
2094 Other than the QXRI, the resolver MUST use the same resolution query parameters as
2095 the original query. If the Resolution Output Format is an XRDS document, the resolver
2096 MUST embed a nested XRDS document containing an XRD representing the Ref as
2097 defined in section 12.5.
- 2098 5. If resolution of an `xrd:Ref` element fails during the authority resolution phase of the
2099 original resolution query, or if resolution of an `xrd:Ref` element fails during the optional
2100 service endpoint selection phase OR the final XRD does not contain the requested
2101 service endpoint, then the resolver MUST record the nested XRDS document as far as
2102 resolution was successful, including the relevant status codes for each XRD as specified
2103 in section 15. The resolver MUST then select the next highest priority `xrd:Ref` element
2104 as specified above and repeat rule 5. For more details, see section 12.6, *Recursion and*
2105 *Backtracking*.
- 2106 6. If resolution of all `xrd:Ref` elements in the XRD or SEP originating Ref processing fails,
2107 the resolver MUST stop and return a 26x error in the XRD containing the Ref or as a
2108 plain text error message as specified in section 15. The resolver MUST NOT try any
2109 other SEPs even if multiple SEPs were selected as specified in section 13.
- 2110 7. If resolution of an `xrd:Ref` element succeeds and `cid=true`, the resolver MUST
2111 perform CanonicalID verification across all XRDs in the nested XRDS document as
2112 specified in section 14.3. Note that each set of XRDs in each new nested XRDS
2113 document produced as a result of Redirect or Ref processing constitutes its own
2114 CanonicalID verification chain. *CanonicalID verification never crosses between XRDS*
2115 *documents*. See section 12.5 for examples.
- 2116 8. If resolution of an `xrd:Ref` element succeeds and the final XRD contains the service
2117 endpoint(s) necessary to continue or complete the original resolution query, further
2118 authority resolution or service endpoint selection MUST continue based on the final XRD.

Comment [DSR13]: Typo caught
by Wil Tan.

Deleted: be

2119 12.5 Nested XRDS Documents

2120 Processing of a Redirect or Ref ALWAYS produces a new XRDS document that describes the
2121 Redirect or Ref that was followed, even if the result was an error. If the final requested Resolution
2122 Output Format is NOT an XRDS document, this new XRDS document is only needed to obtain
2123 the metadata necessary to continue or complete resolution. However, if the final requested
2124 Resolution Output Format is an XRDS document, each XRDS document produced as a result of
2125 Redirect or Ref processing MUST be nested inside the outer XRDS document immediately
2126 following the `xrd:XRD` element containing the `xrd:Redirect` or `xrd:Ref` element being
2127 followed. If more than one Redirect or Ref element is resolved due to an error, the corresponding
2128 nested XRDS documents MUST be included in the same order as the Redirect or Ref elements
2129 that were followed to produce them.

2130 Each new XRDS document is a recursive authority resolution call and MUST conform to all
2131 authority resolution requirements. In addition, the following rules apply:

- 2132 • For a Redirect, the `xrds:XRDS/@redirect` attribute of the nested XRDS document MUST
2133 contain the fully-constructed HTTP(S) URI it describes as specified in section 12.3.
- 2134 • For a Ref, the `xrds:XRDS/@ref` attribute of the nested XRDS document MUST contain the
2135 exact value of the `xrd:XRD/xrd:Ref` element it describes.

2136 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
2137 original query identifier even if resolution traverses multiple Redirects or Refs, and even if errors
2138 were encountered. Like the outer XRDS document, nested XRDS documents MUST NOT include
2139 an XRD for the community root subsegment because this is part of the configuration of the
2140 resolver.

2141 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD with an
2142 `xml:id` attribute value matching the `xml:id` attribute value of any previous XRD in the chain of
2143 resolution requests beginning with the original QXRI, the resolver MUST replace this XRD with an
2144 empty XRD element. The resolver MUST set this empty element's `idref` attribute value to the
2145 value of the `xml:id` attribute of the matched XRD element. This prevents conflicting `xml:id`
2146 values.

2147 12.5.1 Redirect Examples

2148 Example #1:

2149 In this example the original query identifier is `xri://@a`. The first XRD contains an XRD-level
2150 Redirect to `http://a.example.com/`. The elements and attributes specific to Redirect
2151 processing are shown in **bold**. CanonicalIDs are included to illustrate the synonym verification
2152 rule in section 12.3.

```
2153 <XRDS xmlns="xri://$xrds" ref="xri://@a">
2154   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2155     <Query>*a</Query>
2156     <ProviderID>xri://@</ProviderID>
2157     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2158     <Redirect>http://a.example.com/</Redirect>
2159     ...
2160   </XRD>
2161   <XRDS redirect="http://a.example.com/">
2162     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2163       <ProviderID>xri://@</ProviderID>
2164       <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1 CID #1
2165       ...
2166     <Service>
2167       <Type>http://openid.net/signon/1.0</Type>
2168       <URI>http://openid.example.com/</URI>
```

2169
2170
2171
2172

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

2173 **Example #2:**

2174 In this example the original query identifier is `xri://a*b*c`. The second XRD contains a SEP-
2175 level Redirect in its authority resolution SEP to `http://other.example.com/`. Note that
2176 because authority resolution is not complete when this Redirect is encountered, it continues in the
2177 outer XRDS after the nested XRDS representing the Redirect is complete. Again, CanonicalIDs
2178 are included to illustrate the synonym verification rule.

2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222

```
<XRDS xmlns="xri://$xrds" ref="xri://a*b*c">  
  <XRD xmlns="xri://$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*($v*2.0)</Type>  
      <Redirect>http://other.example.com</Redirect>  
    </Service>  
  </XRD>  
<XRDS redirect="http://other.example.com">  
  <XRD xmlns="xri://$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>
```

2223 **Example #3:**

2224 In this example the original query identifier is again `xri://@a*b*c`. This time the final XRD
2225 contains a SEP-level Redirect to `http://other.example.com/`. Because authority resolution
2226 is complete, the outer XRDS ends with a nested XRDs representing the SEP-level Redirect.

```
2227 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2228   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2229     <Query>*a</Query>
2230     <ProviderID>xri://@</ProviderID>
2231     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2232     ...
2233     <Service>
2234       <Type>xri://$res*auth*($v*2.0)</Type>
2235       <URI>http://a.example.com/</URI>
2236     </Service>
2237   </XRD>
2238   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2239     <Query>*b</Query>
2240     <ProviderID>xri://@!1</ProviderID>
2241     <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2
2242     ...
2243     <Service>
2244       <Type>xri://$res*auth*($v*2.0)</Type>
2245       <URI>http://b.example.com/</URI>
2246     </Service>
2247   </XRD>
2248   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2249     <Query>*c</Query>
2250     <ProviderID>xri://@!1!2</ProviderID>
2251     <CanonicalID>xri://@!1!2!3</CanonicalID> ;XRDS #1 CID #3
2252     ...
2253     <Service>
2254       <Type>http://openid.net/signon/1.0</Type>
2255       <Redirect>http://r.example.com/openid</Redirect>
2256     </Service>
2257   </XRD>
2258   <XRDS redirect="http://r.example.com/openid">
2259     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2260       <ProviderID>xri://@!1!2</ProviderID>
2261       <CanonicalID>xri://@!1!2!3</CanonicalID> ;SAME AS XRDS #1 CID #3
2262       ...
2263       <Service>
2264         <Type>http://openid.net/signon/1.0</Type>
2265         <URI>http://openid.example.com/</URI>
2266       </Service>
2267     </XRD>
2268   </XRDS>
2269 </XRDS>
```

2270

Example #4:

2271

In this final example the query identifier is `xri://@a*b`. The first XRD contains an XRD-level Redirect to `http://a.example.com/`, and this XRDS document in turn contains a second redirect to `http://b.example.com/`. Chaining redirects in this manner is NOT RECOMMENDED but is shown here to clarify how XRDS document nesting works.

2272

2273

2274

Comment [DSR14]: Added in Revision 02 per suggestion from Wil Tan to clarify how double-nested XRDS documents work.

2275

```

2276 <XRDS xmlns="xri://$xrds" ref="xri://@a*b">
2277   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2278     <Query>*a</Query>
2279     <ProviderID>xri://@</ProviderID>
2280     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2281     <Redirect>http://a.example.com/</Redirect>
2282     ...
2283   </XRD>
2284   <XRDS redirect="http://a.example.com/">
2285     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2286       <ProviderID>xri://@</ProviderID>
2287       <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1 CID #1
2288       <Redirect>http://b.example.com/</Redirect>
2289       ...
2290     </XRD>
2291     <XRDS redirect="http://b.example.com/">
2292       <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2293         <ProviderID>xri://@</ProviderID>
2294         <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1 CID #1
2295         ...
2296         <Service>
2297           <Type>xri://$res*auth*($v*2.0)</Type>
2298           <URI>http://b.example.com/</URI>
2299         </Service>
2300       </XRD>
2301     </XRDS>
2302   </XRDS>
2303   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2304     <Query>*b</Query>
2305     <ProviderID>xri://@!1</ProviderID>
2306     <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2
2307     ...
2308     <Service>
2309       <Type>xri://$res*auth*($v*2.0)</Type>
2310       <URI>http://b.example.com/</URI>
2311     </Service>
2312   </XRD>
2313 </XRDS>

```


2314 12.5.2 Ref Examples

2315 Example #1:

2316 In this example the original query identifier is `xri://@a`. The first XRD contains an XRD-level
2317 Ref to `xri://@x*y`. The CanonicalID values are included to illustrate the CanonicalID
2318 verification rules in section 14.3.

```
2319 <XRDS xmlns="xri://$xrds" ref="xri://@a">
2320   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2321     <Query>*a</Query>
2322     <ProviderID>xri://@</ProviderID>
2323     <CanonicalID>xri://!1</CanonicalID>      ;XRDS #1 CID #1
2324     <Ref>xri://@x*y</Ref>
2325   </XRD>
2326   <XRDS ref="xri://@x*y">
2327     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2328       <Query>*x</Query>
2329       <ProviderID>xri://@</ProviderID>
2330       <CanonicalID>xri://!7</CanonicalID>    ;XRDS #2 CID #1
2331       ...
2332       <Service>
2333         <Type>xri://$res*auth*($v*2.0)</Type>
2334         <URI>http://x.example.com/</URI>
2335       </Service>
2336     </XRD>
2337     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2338       <Query>*y</Query>
2339       <ProviderID>xri://!7</ProviderID>
2340       <CanonicalID>xri://!7!8</CanonicalID> ;XRDS #2 CID #2
2341       ...
2342       <Service>
2343         <Type>xri://$res*auth*($v*2.0)</Type>
2344         <URI>http://y.example.com/</URI>
2345       </Service>
2346       <Service>
2347         <Type>http://openid.net/signon/1.0</Type>
2348         <URI>http://openid.example.com/</URI>
2349       </Service>
2350     </XRD>
2351   </XRDS>
2352 </XRDS>
```

2353 Example #2:

2354 In this example the original query identifier is `xri://@a*b*c`. The second XRD contains a SEP-
2355 level Ref in its authority resolution SEP to `xri://@x*y`. Note that because authority resolution is
2356 not complete when this Ref is encountered, it continues in the outer XRDS after the nested XRDS
2357 representing the Ref. *Note especially how the CanonicalIDs progress to satisfy the CanonicalID*
2358 *verification rules specified in section 14.3.*

```
2359 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2360   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2361     <Query>*a</Query>
2362     <ProviderID>xri://@</ProviderID>
2363     <CanonicalID>xri://!1</CanonicalID>      ;XRDS #1 CID #1
2364     ...
2365     <Service>
2366       <Type>xri://$res*auth*($v*2.0)</Type>
2367       <URI>http://a.example.com/</URI>
```

```

2368     </Service>
2369 </XRD>
2370 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2371   <Query>*b</Query>
2372   <ProviderID>xri://@!1</ProviderID>
2373   <CanonicalID>xri://@!1!2</CanonicalID>           ;XRDS #1 CID #2
2374   ...
2375   <Service>
2376     <Type>xri://$res*auth*($v*2.0)</Type>
2377     <Ref>xri://@x*y</Ref>
2378   </Service>
2379 </XRD>
2380 <XRDS ref="xri://@x*y">
2381   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2382     <Query>*x</Query>
2383     <ProviderID>xri://@</ProviderID>
2384     <CanonicalID>xri://@!7</CanonicalID>           ;XRDS #2 CID #1
2385     ...
2386     <Service>
2387       <Type>xri://$res*auth*($v*2.0)</Type>
2388       <URI>http://x.example.com/</URI>
2389     </Service>
2390   </XRD>
2391   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2392     <Query>*y</Query>
2393     <ProviderID>xri://@!7</ProviderID>
2394     <CanonicalID>xri://@!7!8</CanonicalID>         ;XRDS #2 CID #2
2395     ...
2396     <Service>
2397       <Type>xri://$res*auth*($v*2.0)</Type>
2398       <URI>http://y.example.com/</URI>
2399     </Service>
2400   </XRD>
2401 </XRDS>
2402 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2403   <Query>*c</Query>
2404   <ProviderID>xri://@!1!2</ProviderID>
2405   <CanonicalID>xri://@!1!2!3</CanonicalID>         ;XRDS #1 CID #3 IS
2406   CHILD OF XRDS #1 CID #2
2407   ...
2408   <Service>
2409     ...final service endpoints described here...
2410   </Service>
2411 </XRD>
2412 </XRDS>

```

2413 **Example #3:**

2414 In this example the original query identifier is again xri://@a*b*c. This time the final XRD
2415 contains a SEP-level Ref to xri://@x*y. Because authority resolution is complete, the outer
2416 XRDS ends with a nested XRDS representing the SEP-level Ref.

```

2417 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2418   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2419     <Query>*a</Query>
2420     <ProviderID>xri://@</ProviderID>
2421     <CanonicalID>xri://@!1</CanonicalID>           ;XRDS #1 CID #1
2422     ...
2423     <Service>
2424       <Type>xri://$res*auth*($v*2.0)</Type>
2425       <URI>http://a.example.com/</URI>
2426     </Service>

```

```

2427 </XRD>
2428 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2429   <Query>*b</Query>
2430   <ProviderID>xri://@!1</ProviderID>
2431   <CanonicalID>xri://@!1!2</CanonicalID>           ;XRDS #1 CID #2
2432   ...
2433   <Service>
2434     <Type>xri://$res*auth*($v*2.0)</Type>
2435     <URI>http://a.example.com/</URI>
2436   </Service>
2437 </XRD>
2438 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2439   <Query>*c</Query>
2440   <ProviderID>xri://@!1!2</ProviderID>
2441   <CanonicalID>xri://@!1!2!3</CanonicalID>       ;XRDS #1 CID #3
2442   ...
2443   <Service>
2444     <Type>http://openid.net/signon/1.0</Type>
2445     <Ref>xri://@x*y</Ref>
2446   </Service>
2447 </XRD>
2448 <XRDS ref="xri://@x*y">
2449   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2450     <Query>*x</Query>
2451     <ProviderID>xri://@</ProviderID>
2452     <CanonicalID>xri://@!7</CanonicalID>         ;XRDS #2 CID #1
2453     ...
2454     <Service>
2455       <Type>xri://$res*auth*($v*2.0)</Type>
2456       <URI>http://x.example.com/</URI>
2457     </Service>
2458   </XRD>
2459   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2460     <Query>*y</Query>
2461     <ProviderID>xri://@!7</ProviderID>
2462     <CanonicalID>xri://@!7!8</CanonicalID>       ;XRDS #2 CID #2
2463     ...
2464     <Service>
2465       <Type>xri://$res*auth*($v*2.0)</Type>
2466       <URI>http://y.example.com/</URI>
2467     </Service>
2468     <Service>
2469       <Type>http://openid.net/signon/1.0</Type>
2470       <URI>http://openid.example.com/</URI>
2471     </Service>
2472   </XRD>
2473 </XRDS>
2474 </XRDS>

```

2475

12.6 Recursion and Backtracking

2476

Redirect and Ref processing triggers recursive calls to authority resolution that produce nested XRDS documents. This recursion can continue to any depth, i.e., a Redirect may contain another Redirect or a Ref, and a Ref may contain another Ref or a Redirect. To avoid confusion, either in resolver implementations or in XRDS documents, it is important to clarify the “backtracking” rules. The following should be read in conjunction with the flowcharts in Figure 2, Figure 5, Figure 7, and Figure 8.

2482

- *Separation of phases.* Redirect and Ref processing invoked during the authority resolution phase is separate and distinct from Redirect and Ref processing invoked during the optional service endpoint selection phase (see Figure 2). Redirect or Ref processing during the former MUST successfully complete authority resolution or else return an error. Redirect or Ref processing during the latter MUST successfully locate the requested service endpoint or else return an error, i.e., it MUST NOT backtrack into the authority resolution phase.

2488

- *First recursion point.* The first time a resolver encounters a Redirect or a Ref within a phase is called the *first recursion point*. There MUST be at most one first recursion point during the authority resolution phase and at most one first recursion point during the optional service endpoint selection phase. During the authority resolution phase, the first recursion point MAY be either an XRD or a service endpoint (SEP). During the optional service endpoint selection phase, the first recursion point MUST be a SEP.

2494

- *Priority order.* As specified in sections 12.3 and 12.4, once a resolver reaches a first recursion point during the authority resolution stage, it MUST process Redirects or Refs in priority order until either it successfully completes authority resolution (and the final XRD does not contain an XRD-level Redirect or Ref), or until all Redirects or Refs have failed. Similarly, once a resolver reaches a first recursion point during the optional service endpoint selection phase, it MUST process Redirect or Ref in priority order until either it successfully locates the requested SEP (and that SEP does not contain a Redirect or Ref), or until all Redirects or Refs have failed.

2502

- *Next recursion point.* If a Redirect or Ref leads to another Redirect or Ref, this is called the *next recursion point*. The same rules apply to the next recursion point as apply to the first recursion point, except that if all attempts to resolve a Redirect or Ref at a next recursion point fail, the resolver MUST return to the previous recursion point and continue trying any untried Redirects or Refs until either it is successful or all Redirects or Refs have failed.

2507

- *Termination.* If the resolver returns to the first recursion point and all of its Redirects or Refs have failed, the resolver MUST stop and return an error.

2509

To avoid excessive recursion and inefficient resolution responses, XRDS authors are RECOMMENDED to use as few Redirects or Refs in a resolution chain as possible.

2510

Deleted: ny
Deleted: point completely fail
Deleted: s
Comment [DSR15]: Improved wording suggested by Wil Tan.

2511

13 Service Endpoint Selection

2512

The second phase of XRI resolution is called *service endpoint selection*. As noted in Figure 2, this

2513

phase is invoked automatically for each iteration of authority resolution after the first in order to

2514

select the Next Authority Resolution Service Endpoint as defined in section 9.1.9. It is also

2515

performed after authority resolution is complete if optional service endpoint selection is

2516

requested.

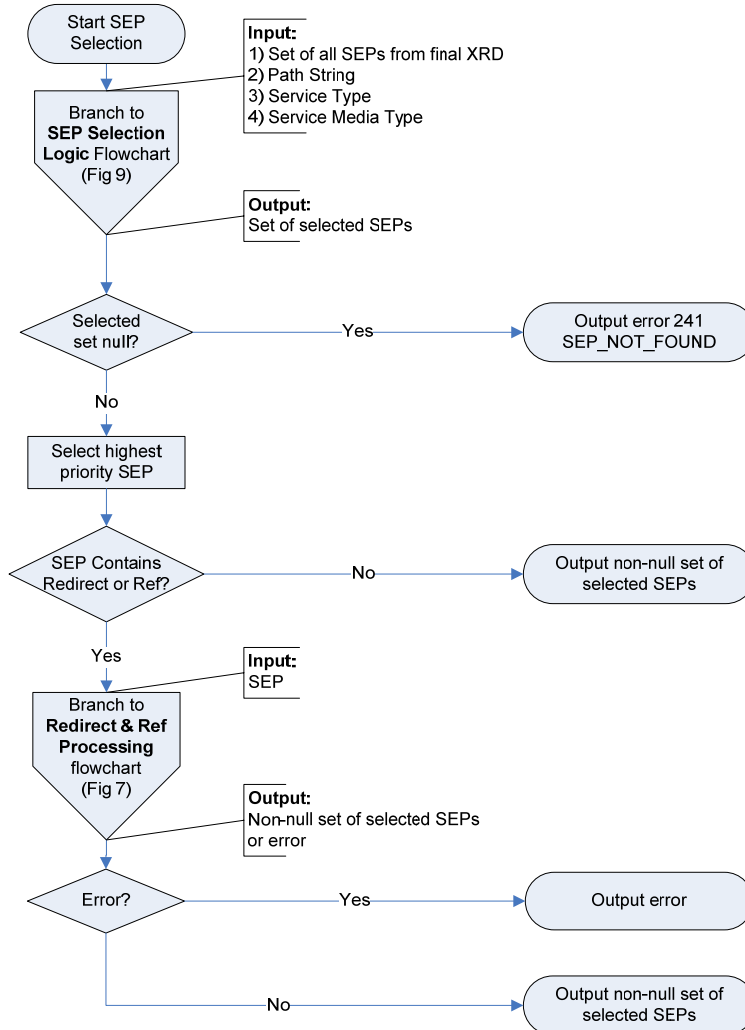
2517

13.1 Processing Rules

2518

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

Comment [DSR16]: This flowchart revised for improved clarity about outputs and recording of XRD and XRDS documents.



2519

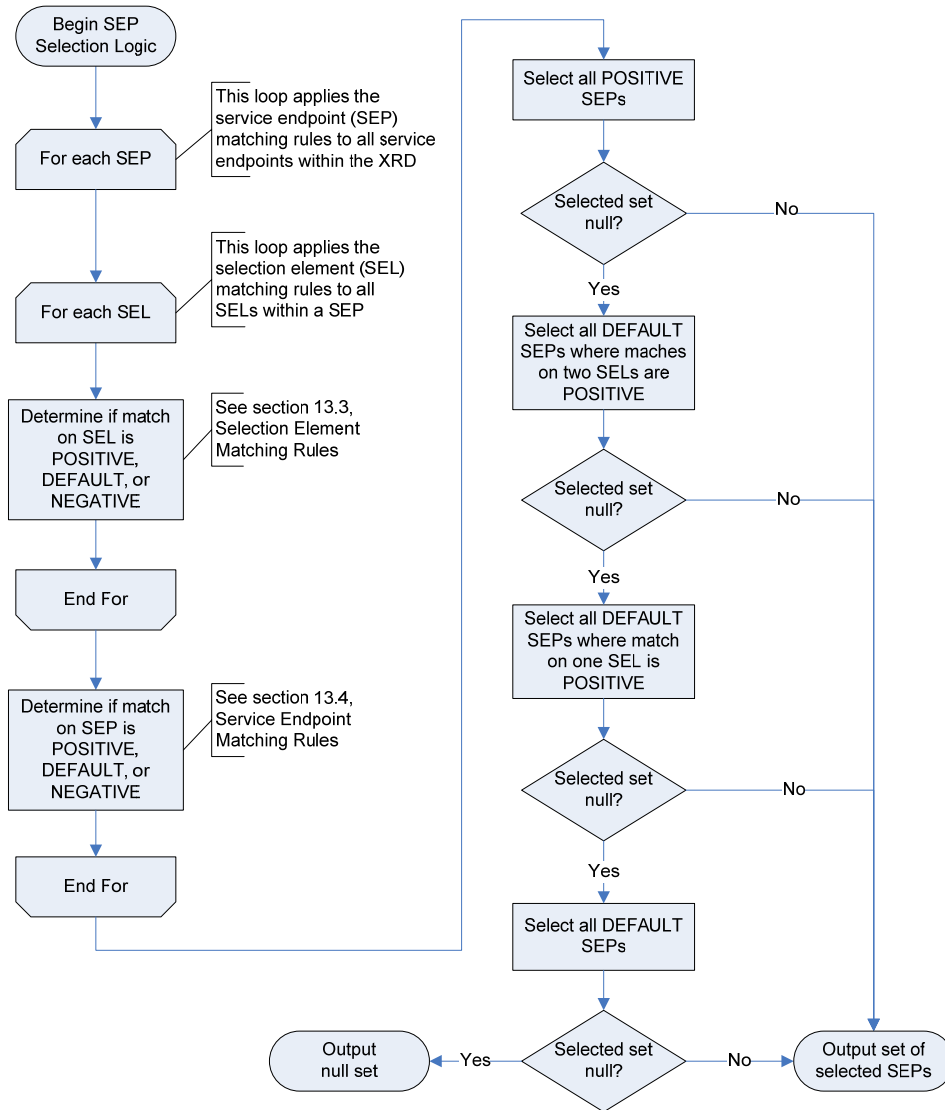
2520

Figure 8: Service endpoint (SEP) selection flowchart.

- 2521 Following are the normative rules for the overall service endpoint selection process:
- 2522 1. The inputs for service endpoint selection are defined in Table 8.
- 2523 2. For the set of all service endpoints (`xrd:XRD/xrd:Service` elements) in the XRD,
2524 service endpoint selection **MUST** follow the logic defined in section 13.2. The output of
2525 this process **MUST** be either the null set or a selected set of one or more service
2526 endpoints.
- 2527 3. If, after applying the service endpoint selection logic, the selected set is null, this function
2528 **MUST** return the error 241 `SEP_NOT_FOUND`.
- 2529 4. If, after applying the service endpoint selection logic, the selected set is not null and the
2530 highest priority selected service endpoint contains an
2531 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
2532 element, it **MUST** first be processed as specified in section 12. This is a recursive call
2533 that will produce a nested XRDS document as defined in section 12.5.

2534 **13.2 Service Endpoint Selection Logic**

2535 Selection of service endpoints (SEPs) within an XRD is managed using service endpoint
 2536 selection elements (SEs). As shown in Figure 9 (non-normative), the selection process first
 2537 applies SEL matching rules (section 13.3), followed by SEP matching rules (section 13.4), to the
 2538 set of all SEPs in the XRD. It then applies SEP selection rules (section 13.5) to determine the
 2539 final output.



2540
 2541 *Figure 9: Service endpoint (SEP) selection logic flowchart.*

2542 The following sections provide the normative rules for each section of this flowchart.

2543 **13.3 Selection Element Matching Rules**

2544 The first set of rules govern the matching of selection elements.

2545 **13.3.1 Selection Element Match Options**

2546 As defined in section 4.2.6, there are three categories of service endpoint selection elements:
2547 `xrd:Type`, `xrd:Path`, and `xrd:MediaType`. Within each service endpoint, there is a match
2548 option for each of the three categories of selection elements. Matches are tri-state: the three
2549 options and their corresponding precedence order are defined in Table 24:

Match Option	Match Condition	Precedence
POSITIVE	A successful match based on the value of the <code>match</code> attribute as defined in 13.3.2 OR a successful match based the contents of the selection element as defined in sections 13.3.6 - 13.3.8.	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 13.3.3.	0
NEGATIVE	The selection element does not satisfy either condition above.	-1

2550 *Table 24: Match options for selection elements.*

2551 The Precedence order is used in the Multiple Selection Element Matching Rule (section 13.3.5).

2552 **IMPORTANT:** Failure of a POSITIVE match does not necessarily mean a NEGATIVE match; it
2553 may still qualify as a DEFAULT match.

2554 **13.3.2 The Match Attribute**

2555 All three service endpoint selection elements accept the optional `match` attribute. This attribute
2556 gives XRDS authors precise control over selection of SEPs based on the QXRI and other service
2557 endpoint selection parameters. An enumerated list of the values for the `match` attribute is defined
2558 in Table 25. If the `match` attribute is present with one of these values, the contents of the
2559 selection element **MUST** be ignored, and the corresponding matching rule **MUST** be applied. If
2560 the `match` attribute is absent or has any other value, the rules in this section do not apply.

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> subparameter is set to TRUE 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.

2561 *Table 25: Enumerated values of the global match attribute and corresponding matching rules.*

2562 BACKWARDS COMPATIBILITY NOTE: earlier working drafts of this specification included the
2563 values `match="none"` and `match="contents"`. Both are deprecated. The former is no longer
2564 supported and the latter is now the default behaviour of any selection element that does not
2565 include the `match` attribute. Implementers SHOULD accept these values accordingly.

2566 13.3.3 Absent Selection Element Matching Rule

2567 If a service endpoint does not contain at least one instance of a particular category of selection
2568 element, it MUST be considered equivalent to the service endpoint having a DEFAULT match on
2569 that category of selection element UNLESS overridden by a `nodefault_*` parameter as specified
2570 in Table 25.

2571 13.3.4 Empty Selection Element Matching Rule

2572 If a selection element is present in a service endpoint but the element is empty, and if the element
2573 does not contain a `match` attribute, it MUST be considered equivalent to having a `match`
2574 attribute with a value of `null`.

2575 13.3.5 Multiple Selection Element Matching Rule

2576 Each service endpoint has only one match option for each category of selection element.
2577 Therefore if a service endpoint contains more than one instance of the same category of selection
2578 element (i.e., more than one `xrd:Type`, `xrd:Path`, or `xrd:MediaType` element), the match for
2579 that category of selection element MUST be the match for the selection element(s) with the
2580 highest precedence match option as defined in Table 24.

2581 13.3.6 Type Element Matching Rules

2582 The following rules apply to matching the value of the input Service Type parameter with the
2583 contents of a non-empty `xrd:XRD/xrd:Service/xrd:Type` element when its `match` attribute
2584 is absent.

- 2585 1. If the value is an XRI or IRI, it MUST be in URI-normal form as defined in section 4.4.
- 2586 2. Prior to comparison (and only for the purpose of comparison), the values of the Service
2587 Type parameter and the `xrd:XRD/xrd:Service/xrd:Type` element SHOULD be
2588 normalized according to the requirements of their identifier scheme. In particular, if an
2589 XRI, IRI, or URI uses hierarchical syntax and does not include a local part (a path and/or
2590 query component) after the authority component, a trailing forward slash after the
2591 authority component MUST NOT be considered significant in comparisons. In all other
2592 cases, a trailing forward slash MUST be considered significant in comparisons unless this
2593 rule is overridden by scheme-specific comparison rules.
- 2594 3. To result in a POSITIVE match on this selection element, the values MUST be equivalent
2595 according to the equivalence rules of the applicable identifier scheme. Any other result is
2596 a NEGATIVE match on this selection element.

2597 As a best practice, service architects SHOULD assign identifiers for service types that are in URI-
2598 normal form, do not require further normalization, and are easy to match.

2599 13.3.7 Path Element Matching Rules

2600 The following rules apply to matching the value of the input Path String (the path portion of the
2601 QXRI as defined in section 8.1.1) with the contents of a non-empty
2602 `xrd:XRD/xrd:Service/xrd:Path` element when its `match` attribute is absent.

- 2603 1. If the value is a relative XRI or an IRI it MUST be in URI-normal form as defined in
2604 section 4.4.
- 2605 2. Prior to comparison, the leading forward slash separating an XRI authority component
2606 from the path component MUST be prepended to the Path String. Any subsequent
2607 forward slash, including trailing forward slashes, MUST be significant in comparisons.
- 2608 3. The contents of the `xrd:XRD/xrd:Service/xrd:Path` element SHOULD include the
2609 leading forward slash separating the XRI authority component from the path. If it does
2610 not, one MUST be prepended prior to comparison.
- 2611 4. Equivalence comparison SHOULD be performed using Caseless Matching as defined in
2612 section 3.13 of **[Unicode]**.
- 2613 5. To result in a POSITIVE match on this selection element, the value of the Path String
2614 MUST be a *subsegment stem match* with the contents of the
2615 `xrd:XRD/xrd:Service/xrd:Path` element. A subsegment stem match is defined as
2616 the entire Path String being character-for-character equivalent with any continuous
2617 sequence of subsegments or segments (including empty subsegments and empty
2618 segments) in the contents of the Path element beginning from the most significant
2619 (leftmost) subsegment. Subsegments and segments are formally defined in **[XRI Syntax]**.
2620 Any other result MUST be a NEGATIVE match on this selection element.

2621 Examples of this rule are shown in Table 26.

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

2622 Table 26: Examples of applying the Path element matching rules.

2623 13.3.8 MediaType Element Matching Rules

2624 The following rules apply to matching the value of the input Service Media Type parameter with
2625 the contents of of a non-empty `xrd:XRD/xrd:Service/xrd:MediaType` element when its
2626 `match` attribute is absent.

- 2627 1. The values of the Service Media Type parameter and the `xrd:MediaType` element
2628 SHOULD be normalized according to the rules for media types in section 3.7 of
2629 [RFC2616] prior to input. (The rules are that media type and media type parameter
2630 names are case-insensitive, but parameter values may or may not be case-sensitive
2631 depending on the semantics of the parameter name. XRI Resolution Output Format
2632 parameters and subparameters are all case-insensitive.) XRI resolvers MAY perform
2633 normalization of these values but MUST NOT be required to do so.
- 2634 2. To be a POSITIVE match on this selection element, the values MUST be character-for-
2635 character equivalent. Any other result is a NEGATIVE match on this selection element.

2636 13.4 Service Endpoint Matching Rules

2637 The next set of matching rules govern the matching of service endpoints based on the matches of
2638 the selection elements they contain.

2639 13.4.1 Service Endpoint Match Options

2640 For each service endpoint in an XRD, there are three match options as defined in Table 27:

Match Option	Condition
POSITIVE	Meets the Select Attribute Match Rule (section 13.4.2) or the All Positive Match Rule (section 13.4.3).
DEFAULT	Meets the Default Match Rule (section 13.4.4).
NEGATIVE	The service endpoint does not satisfy either condition above.

2641 *Table 27: Match options for service endpoints.*

2642 13.4.2 Select Attribute Match Rule

2643 All three service endpoint selection elements accept the optional `select` attribute. This attribute
2644 is a Boolean value used to govern matching of the containing service endpoint according to the
2645 following rule. If service endpoint contains a selection element with a POSITIVE match as defined
2646 in section 13.3, and the value of this selection element's `select` attribute is TRUE, the service
2647 endpoint automatically MUST be a POSITIVE match, i.e., all other selection elements for this
2648 service endpoint MUST be ignored.

2649 13.4.3 All Positive Match Rule

2650 If a service endpoint has a POSITIVE match on all three categories of selection elements
2651 (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) as defined in section 13.3, the service endpoint
2652 MUST be a POSITIVE match. If even one of the three selection element match types is not
2653 POSITIVE, this rule fails.

2654 13.4.4 Default Match Rule

2655 If a service endpoint fails the Select Attribute Match Rule and the All Positive Match Rule, but
2656 none of the three categories of selection elements has a NEGATIVE match as defined in section
2657 13.3, the service endpoint MUST be a DEFAULT match.

2658 13.5 Service Endpoint Selection Rules

2659 The final set of rules governs the selection of service endpoints based on their matches.

2660 13.5.1 Positive Match Rule

2661 After applying the matching rules to service endpoints in section 13.4, all service endpoints that
2662 have a POSITIVE match MUST be selected. Only if there are no service endpoints with a
2663 POSITIVE match is the Default Match Rule invoked.

2664 13.5.2 Default Match Rule

2665 If the Positive Match Rule above fails, then the service endpoints with a DEFAULT match that
2666 have the highest number of POSITIVE matches on each category of selection element MUST be
2667 selected. This means:

- 2668 1. The service endpoints in the DEFAULT set that have two POSITIVE selection element
2669 matches MUST be selected.
- 2670 2. If the previous set is empty, the service endpoints in the DEFAULT set that have one
2671 POSITIVE selection element match MUST be selected.
- 2672 3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.
- 2673 4. If the previous set is empty, no service endpoint is selected and the return set is null.

2674 13.6 Pseudocode

2675 The following pseudocode provides a precise description of the service endpoint selection logic.
2676 The pseudocode is normative, however if there is a conflict between it and the rules stated in the
2677 preceding sections, the preceding sections shall prevail.

2678 The pseudocode uses nine Boolean flags to record the match state for each category of selection
2679 element (SEL) in a service endpoint (SEP):

- 2680 • `Positive.x` (where `x = Type, Path, or MediaType`)
- 2681 • `Default.x` (where `x = Type, Path, or MediaType`)
- 2682 • `Present.x` (where `x = Type, Path, or MediaType`)

2683 The variable `Nodefault.x` refers to the value of the `nodefault_t` (Type), `nodefault_p`
2684 (Path), and `nodefault_m` (MediaType) subparameters as explained in Table 25.

2685 Note that the complete set of nine SEL match flags is needed for each SEP. The pseudocode first
2686 does a loop through all SEPs in the XRD to:

- 2687 1. Set the SEL match flags according to the rules specified in section 13.3;
- 2688 2. Process the SEL match flags to apply the SEP matching rules specified in section 13.4;
- 2689 3. Apply the positive SEP selection rule specified in section 13.5.1.

2690 After this loop is complete, the pseudocode tests to see if default SEP selection processing is
2691 required. If so, it performs a second loop applying the default SEP selection rules specified in
2692 section 13.5.2.

2693 NOTE: In this pseudocode, when the words POSITIVE, DEFAULT, or NEGATIVE appear in
2694 UPPERCASE, they refer to the SEL match type or SEP match type as defined in Table 24 and
2695 Table 27. When they appear in First Letter Caps, they refer to the Boolean flags defined above.

Comment [DSR17]: Rewritten per Gabe Wachob's suggestion to make the pseudocode clearer.

Comment [DSR18]: Added per Gabe Wachob's suggestion to make the pseudocode clearer.

2696

```
2697 FOR EACH SEP
2698 CREATE set of nine SEL match flags (see text above)
2699 SET all flags to FALSE
2700 FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
2701 SET Present.x=TRUE
2702 IF match type on this SEL is POSITIVE
2703     IF select="true" ;see 13.4.2
2704         ADD SEP TO SELECTED SET
2705         NEXT SEP
2706     ELSE
2707         SET Positive.x=TRUE
2708     ENDIF
2709 ELSEIF match="default" ;see 13.3.2
2710     IF Positive.x != TRUE AND ;see 13.3.5
2711     Nodefault.x != TRUE ;see 13.3.2
2712         SET Default.x=TRUE
2713     ENDIF
2714 ENDIF
2715 ENDFOR
2716 FOR EACH category x (where x=Type, Path, or Mediatype)
2717     IF Present.x=FALSE ;see 13.3.3
2718     IF Nodefault.x != TRUE ;see 13.3.2
2719     SET Default.x=TRUE
2720     ENDIF
2721 ENDIF
2722 ENDFOR
2723 IF Positive.Type=TRUE AND
2724     Positive.Path=TRUE AND
2725     Positive.Mediatype=TRUE ;see 13.4.3
2726     ADD SEP TO SELECTED SET
2727     NEXT SEP
2728 ELSEIF SELECTED SET != EMPTY ;see 13.5.1
2729     NEXT SEP
2730 ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
2731     (Positive.Path=TRUE OR Default.Path=TRUE) AND
2732     (Positive.Mediatype=TRUE OR Default.Mediatype=TRUE)
2733     ADD SEP TO DEFAULT SET ;see 13.4.4
2734 ENDIF
2735 ENDFOR
2736 IF SELECTED SET = EMPTY
2737     FOR EACH SEP IN DEFAULT SET ;see 13.5.2
2738     IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
2739     (Positive.Type=TRUE AND Positive.Mediatype=TRUE) OR
2740     (Positive.Path=TRUE AND Positive.Mediatype=TRUE)
2741     ADD SEP TO SELECTED SET
2742     ENDIF
2743 ENDFOR
2744 IF SELECTED SET = EMPTY
2745     FOR EACH SEP IN DEFAULT SET ;see 13.5.2
2746     IF Positive.Type=TRUE OR
2747     Positive.Path=TRUE OR
2748     Positive.Mediatype=TRUE
2749     ADD SEP TO SELECTED SET
2750     ENDIF
2751 ENDFOR
2752 ENDIF
2753 ENDIF
2754 IF SELECTED SET != EMPTY
2755     RETURN SELECTED SET
2756 ELSE
2757     RETURN DEFAULT SET
2758 ENDIF
```

Comment [DSR19]: Changed so this variable appears the same as other pseudocode variables per Gabe Wachob's suggestion.

Comment [DSR20]: Explicit FOR loop added for consistency per Gabe Wachob's suggestion.

2759 **13.7 Construction of Service Endpoint URIs**

2760 The final step in the service endpoint selection process is construction of the service endpoint
2761 URI(s). This step is necessary if either:

- 2762 • The resolution output format is a URI List.
2763 • Automatic URI construction is requested using the `uric` parameter.

2764 **13.7.1 The append Attribute**

2765 The `append` attribute of a `xrd:XRD/xrd:Service/xrd:URI` element is used to specify how
2766 the final URI is constructed. The values of this attribute are shown in Table 28.

Value	Component of QXRI to Append
none	None. This is the default if the <code>append</code> attribute is absent
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>including the leading forward slash</i> b) If only a query is present, the Query String <i>including the leading question mark</i> c) If both a path and a query are present, the entire combination of the Path String <i>including the leading forward slash</i> and the Query String <i>plus the leading question mark</i> Note that as defined in section 8.1.1, a fragment is never part of a QXRI.
authority	Authority String only (including the community root subsegment) <i>not including the trailing forward slash</i>
path	Path String <i>including the leading forward slash</i>
query	Query String <i>including the leading question mark</i>
qxri	Entire QXRI

2767 Table 28: Values of the `append` attribute and the corresponding QXRI component to append.

2768 If the `append` attribute is absent, the default value is `none`. Following are the rules for
2769 construction of the final service endpoint URI based on the value of the `append` attribute.

2770 **IMPORTANT:** Implementers must follow these rules exactly in order to give XRDS authors
2771 precise control over construction of service endpoint URIs.

- 2772 1. If the value is `none`, the exact contents of the `xrd:URI` element **MUST** be returned
2773 directly without any further processing.
- 2774 2. For any other value, the exact value in URI-normal form of the QXRI component specified
2775 in Table 28, *including any leading delimiter(s) and without any additional escaping or*
2776 *percent encoding* **MUST** be appended directly to the exact contents of the `xrd:URI`
2777 element *including any trailing delimiter(s)*. If the value of the QXRI component specified in
2778 Table 28 consists of only a leading delimiter, then this value **MUST** be appended
2779 according to these rules. If the value of the QXRI component specified in Table 28 is null,
2780 then the contents of the `xrd:URI` element **MUST** be returned directly exactly as if the
2781 value of the `append` attribute was `none`.

2782 3. If any HXRI query parameters for proxy resolution were added to an existing QXRI query
2783 component as defined in section 11.3, these query parameters MUST be removed prior
2784 to performing the append operation as also defined in section 11.3. In particular, if after
2785 removal of these query parameters the QXRI query component consists of only a *string*
2786 of one or more question marks (the delimiting question mark plus zero or more additional
2787 question marks) then *exactly one question mark* MUST also be removed. This preserves
2788 the query component of the original QXRI if it was null or contained only question marks.

2789 **IMPORTANT:** Construction of HTTP(S) URIs for authority resolution service endpoints is defined
2790 in section 9.1.10. Note that this involves an additional step taken after all URI construction steps
2791 specified in this section are complete. In other words, if the URI element of an authority resolution
2792 service endpoint includes an `append` attribute, the Next Authority Resolution Service URI MUST
2793 be fully constructed according to the algorithm in this section before appending the Next Authority
2794 String as defined in section 9.1.10.

2795 **WARNING:** Use of any value of the `append` attribute other than `authority` on the URI element
2796 for an authority resolution service endpoint is NOT RECOMMENDED due to the complexity it
2797 introduces.

2798 13.7.2 The `uric` Parameter

2799 The `uric` subparameter of the Resolution Output Format is used to govern whether a resolver
2800 should perform construction of the URI automatically on behalf of a consuming application.
2801 Following are the processing rules for this parameter:

- 2802 1. If `uric=true`, a resolver MUST apply the URI construction rules specified in section
2803 13.7.1 to each `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD in the
2804 resolution chain. Note that this step is identical to the processing a resolver must perform
2805 to output a URI list.
- 2806 2. The resolver MUST replace the value of each `xrd:XRD/xrd:Service/xrd:URI`
2807 element in the final XRD with the fully constructed URI value.
- 2808 3. The resolver MUST subsequently remove the `append` attribute from each
2809 `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD.
- 2810 4. If `uric=false` or the parameter is absent or empty, a resolver MUST NOT perform any
2811 of the processing specified in this section.

2812

14 Synonym Verification

2813 As described in section 5, a consuming application must be able to verify the security of the
2814 binding between the fully-qualified query identifier (the identifier resolved to an XRDS document)
2815 and any synonyms asserted in the final XRD. This section defines synonym verification rules.

14.1 Redirect Verification

2817 As specified in section 12.3, XRI resolvers MUST verify the synonyms asserted in the XRD
2818 obtained by following a Redirect element. These rules are:

- 2819 1. If resolution of the Redirect succeeds, the resolver MUST first verify that the set of XRD
2820 synonym elements (as specified in section 5.2) contained in the new XRD are *equivalent*
2821 *to or a subset of* those contained in the XRD containing the Redirect.
- 2822 2. Secondly, the resolver MUST verify that the content of each synonym element contained
2823 in the new XRD is exactly equivalent to the content of the corresponding element in the
2824 XRD containing the Redirect.
- 2825 3. If either rule above fails, the resolver MUST stop and return the error 253
2826 REDIRECT_VERIFY_FAILED in the XRD where the error occurred or as a plain text error
2827 message as defined in section 15.

2828 For examples see section 12.5.1.

14.2 EquivID Verification

2830 Although XRI resolvers do not automatically perform EquivID synonym verification, a consuming
2831 application can easily request it using the following steps:

- 2832 1. First request resolution for the original query identifier with CanonicalID verification
2833 enabled (`cid=true`).
- 2834 2. From the final XRD in the resolution chain, select the EquivID for which verification is
2835 desired.
- 2836 3. Request resolution of the EquivID identifier.
- 2837 4. From the final XRD in this second resolution chain, determine if there is either: a) a
2838 `xrd:XRD/xrd:EquivID` element, or b) a `xrd:XRD/xrd:CanonicalEquivID` element
2839 whose value matches the verified CanonicalID of the original query identifier. If there is a
2840 match, the EquivID is verified; otherwise it is not verified.

Example:

- 2842 • Fully-Qualified Query Identifier: `http://example.com/user`
- 2843 • Asserted EquivID: `xri://=!1000.c78d.402a.8824.bf20`

2844 First XRDS (for `http://example.com/user` — simplified for illustration purposes):

```
2845 <XRDS>  
2846 <XRD>  
2847 <EquivID>xri://=!1000.c78d.402a.8824.bf20</EquivID>  
2848 <CanonicalID>http://example.com/user</CanonicalID>  
2849 <Service priority="10">  
2850 ...  
2851 </Service>  
2852 ...  
2853 </XRD>  
2854 </XRDS>
```

2855 Second XRDS (for `xri://=!1000.c78d.402a.8824.bf20`):

```
2856 <XRDS>
2857 <XRD>
2858 <Query>!1000.c78d.402a.8824.bf20</Query>
2859 <ProviderID>xri://=</ProviderID>
2860 <EquivID>http://example.com/user</EquivID>
2861 <CanonicalID>xri://=!1000.c78d.402a.8824.bf20</CanonicalID>
2862 <Service priority="10">
2863   ...
2864 </Service>
2865   ...
2866 </XRD>
2867 </XRDS>
```

2868 The EquivID is verified because the XRD in the second XRDS asserts an EquivID backpointer to
2869 the CanonicalID of the XRD in the first XRDS.

2870 14.3 CanonicalID Verification

2871 XRI resolvers automatically perform verification of CanonicalID and CanonicalEquivID synonyms
2872 unless this function is explicitly turned off using the Resolution Output Format subparameter `cid`.
2873 The following synonym verification MUST be applied by an XRI resolver if `cid=true` or the
2874 parameter is absent or empty, and MUST NOT be applied if `cid=false`.

- 2875 1. If the value of the `xrd:XRD/xrd:CanonicalID` element is an HTTP(S) URI, it MUST
2876 be verified as specified in section 14.3.1.
- 2877 2. If the value of the `xrd:XRD/xrd:CanonicalID` element is an XRI, it MUST be verified
2878 as specified in section 14.3.2.
- 2879 3. If the value of the `xrd:XRD/xrd:CanonicalID` element is any other identifier,
2880 CanonicalID verification fails and the resolver MUST return the CanonicalID verification
2881 status specified in section 14.3.4.
- 2882 4. If CanonicalID verification succeeds but the final XRD in the resolution chain also
2883 contains a `xrd:XRD/xrd:CanonicalEquivID` element, it MUST also be verified as
2884 specified in section 14.3.3, and the resolver MUST return the CanonicalEquivID
2885 verification status as specified in section 14.3.4.
- 2886 5. In all cases, since synonym verification depends on trusting each authority in the
2887 resolution chain, trusted resolution (section 10) SHOULD be used with either
2888 `https=true` or `saml=true` or both to provide additional assurance of the authenticity of
2889 the results.

2890 **IMPORTANT:** There is no guarantee that all XRDS that describe the same target resource will
2891 return the same verified CanonicalID or CanonicalEquivID. Different parent authorities may assert
2892 different CanonicalIDs or CanonicalEquivIDs for the same target resource and all of these may all
2893 be verifiable. In addition, due to Redirect and Ref processing, the verified CanonicalID or
2894 CanonicalEquivID returned for an XRI MAY differ depending on the resolution input parameters.
2895 For example, as described in section 12, a request for a specific service endpoint type may
2896 trigger processing of a Redirect or Ref resulting in a nested XRDS document. The final XRD in
2897 the nested XRDS document may come from a different parent authority and have a different but
2898 still verifiable CanonicalID or CanonicalEquivID.

2899 14.3.1 HTTP(S) URI Verification Rules

2900 To verify that an HTTP(S) URI is a valid CanonicalID synonym for a fully-qualified query identifier
2901 (defined in section 5.1), a resolver MUST verify that all the following tests are successful:

- 2902 1. The fully-qualified query identifier MUST also be an HTTP(S) URI.
- 2903 2. The query identifier MUST be resolved as specified in section 6.
- 2904 3. The asserted CanonicalID synonym MUST be an HTTP(S) URI equivalent to: a) the fully-
2905 qualified query identifier, or b) the fully-qualified query identifier plus a valid fragment as
2906 defined by [RFC3986].

2907 See the example in section 14.3.5.

2908 14.3.2 XRI Verification Rules

2909 To verify that an XRI is a valid CanonicalID synonym for a fully-qualified query identifier (defined
2910 in section 5.1), a resolver MUST verify that all the following tests are successful.

- 2911 1. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID`
2912 element MUST consist of two parts:
 - 2913 1) The value of the `xrd:XRD/xrd:CanonicalID` element for the community root
2914 authority as configured in the XRI resolver or asserted in a self-describing XRD
2915 from the community root authority (or via another equivalent mechanism as
2916 described in section 9.1.6).
 - 2917 2) One additional XRI subsegment as defined in [XRISyntax]. For example, if the
2918 value of the `xrd:XRD/xrd:CanonicalID` element for the community root
2919 authority was `@`, then the following would all be verified values for the
2920 `xrd:XRD/xrd:CanonicalID` element in the first XRD in the resolution chain:
2921 `@!1, @!1234, @!example, @example` (note that `@example` is not
2922 recommended because it is not a persistent identifier).
- 2923 2. For each subsequent XRD in the resolution chain, the value of the
2924 `xrd:XRD/xrd:CanonicalID` element MUST consist of the value the
2925 `xrd:XRD/xrd:CanonicalID` element of the preceding XRD in the same XRDS
2926 document plus one additional XRI subsegment. For example, if the value of the
2927 `xrd:XRD/xrd:CanonicalID` element asserted in an XRD is `@!1!2!3`, then the value
2928 of the `xrd:XRD/xrd:CanonicalID` element in the immediately preceding XRD in the
2929 same XRDS document must be `@!1!2`.
- 2930 3. If Redirect or Ref processing is required during resolution as specified in section 12, the
2931 rules above MUST also apply for each nested XRDS document.

Comment [DSR21]: Revised to remove ProviderID correlation requirement (deemed unnecessary) per suggestion from Wil Tan.

2932 **IMPORTANT:** Each set of XRDs in each new nested XRDS document produced as a result of
2933 Redirect or Ref processing constitutes its own CanonicalID verification chain. *CanonicalID*
2934 *verification never crosses between XRDS documents.* See the examples in section 12.5.

2935 14.3.3 CanonicalEquivID Verification

2936 CanonicalID verification also requires verification of a CanonicalEquivID *only if it is present in the*
2937 *final XRD in the resolution chain.* Since CanonicalEquivID verification typically requires an extra
2938 resolution cycle, restricting automatic verification to the final XRD in the resolution chain ensures
2939 it will add at most one additional resolution cycle.

2940 CanonicalEquivID verification MUST NOT be performed unless CanonicalID verification as
2941 specified in section 14.3 has completed successfully. The resulting value is called the *verified*
2942 *CanonicalID*.

2943 To verify that a CanonicalEquivID is an authorized synonym for a verified CanonicalEquivID, a
2944 resolver MUST verify that either: a) the value of the CanonicalEquivID element is character-by-
2945 character equivalent to the verified CanonicalID (since both appear in the same XRD, all other
2946 normalization rules are waived), or b) that all the following tests are successful:

- 2947 1. The asserted CanonicalEquivID value MUST be a valid HTTP(S) URI or XRI.
- 2948 2. The asserted CanonicalEquivID value MUST resolve successfully to an XRDS document
2949 according to the rules in this specification *using the same resolution parameters as in the*
2950 *original resolution request.*
- 2951 3. The CanonicalID in the final XRD of the resolved XRDS document MUST be verified and
2952 MUST be equivalent to the asserted CanonicalEquivID.
- 2953 4. The final XRD in the resolved XRDS document MUST contain either an EquivID or a
2954 CanonicalEquivID "backpointer" whose value is equivalent to the verified CanonicalID in
2955 the XRD asserting the CanonicalEquivID.

2956 SPECIAL SECURITY CONSIDERATION: See section 5.2.2 regarding the rules for provisioning
2957 of `xrd:XRD/xrd:EquivID` and `xrd:XRD/xrd:CanonicalEquivID` elements in an XRD.

2958 14.3.4 Verification Status Attributes

2959 If CanonicalID verification is performed, an XRI resolver MUST return the CanonicalID and
2960 CanonicalEquivID verification status using an attribute of the `xrd:XRD/xrd:Status` element in
2961 each XRD in the output as follows:

- 2962 1. CanonicalID verification MUST be reported using the `cid` attribute.
- 2963 2. CanonicalEquivID verification MUST be reported using the `ceid` attribute.
- 2964 3. Both attributes accept four enumerated values: `absent` if the element is not present, `off`
2965 if verification is not performed, `verified` if the element is verified, and `failed` if
2966 verification fails.
- 2967 4. The `off` value applies to both elements if CanonicalID verification is not performed
2968 (`cid=false`).
- 2969 5. The `off` value applies to the CanonicalEquivID element in any XRD before the final XRD
2970 if CanonicalID verification is performed (`cid=true`), because a resolver only verifies this
2971 element in the final XRD.
- 2972 6. If `cid=true` and verification of any CanonicalID element fails, *verification of all*
2973 *CanonicalIDs in all subsequent XRDs in the same XRDS document MUST fail.*

2974 From these verification status attributes, a consuming application can confirm on every XRD in
2975 the XRDS document whether the CanonicalID is present and has been verified. In addition, for
2976 the final XRD in the XRDS document, it can confirm whether the CanonicalEquivID element is
2977 present and has been verified.

2978 **14.3.5 Examples**

2979 **Example #1:**

- 2980 • Fully-Qualified Query Identifier: `http://example.com/user`
2981 • Asserted CanonicalID: `http://example.com/user#1234`

2982 XRDS (simplified for illustration purposes):

```
2983 <XRDS ref="http://example.com/user">  
2984 <XRD>  
2985 <CanonicalID>http://example.com/user#1234</CanonicalID>  
2986 <Service priority="10">  
2987 ...  
2988 </Service>  
2989 ...  
2990 </XRD>  
2991 </XRDS>
```

2992 The asserted CanonicalID satisfies the HTTP(S) URI verification rules in section 14.3.1.

2993

2994 **Example #2:**

- 2995 • Fully-Qualified Query Identifier: `=example.name*delegate.name`
2996 • Asserted CanonicalID: `!=1000.62b1.44fd.2855!1234`

2997 XRDS (for `=example.name*delegate.name`):

```
2998 <XRDS ref="xri://example.name*delegate.name">  
2999 <XRD>  
3000 <Query>*example.name</Query>  
3001 <ProviderID>xri://=</ProviderID>  
3002 <LocalID>!1000.62b1.44fd.2855</LocalID>  
3003 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>  
3004 <Service>  
3005 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>  
3006 <Type>xri://$res*auth*($v*2.0)</Type>  
3007 <MediaType>application/xrds+xml</MediaType>  
3008 <URI priority="10">http://resolve.example.com</URI>  
3009 <URI priority="15">http://resolve2.example.com</URI>  
3010 <URI>https://resolve.example.com</URI>  
3011 </Service>  
3012 ...  
3013 </XRD>  
3014 <XRD>  
3015 <Query>*delegate.name</Query>  
3016 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>  
3017 <LocalID>!1234</LocalID>  
3018 <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>  
3019 <Service priority="1">  
3020 ...  
3021 </Service>  
3022 ...  
3023 </XRD>  
3024 </XRDS>
```

3025 The asserted CanonicalID satisfies the XRI verification rules in section 14.3.2.

3026

3027 **Example #3:**

- 3028 • Fully-Qualified Query Identifier: `http://example.com/user`
- 3029 • Asserted CanonicalID: `http://example.com/user`
- 3030 • Asserted CanonicalEquivID: `https://different.example.net/path/user`

3031 First XRDS (for `http://example.com/user`):

```
3032 <XRDS ref="http://example.com/user">
3033 <XRD>
3034 <CanonicalID>http://example.com/user</CanonicalID>
3035 <CanonicalEquivID>
3036 https://different.example.net/path/user
3037 </CanonicalEquivID>
3038 <Service priority="10">
3039 ...
3040 </Service>
3041 ...
3042 </XRD>
3043 </XRDS>
```

3044 Second XRDS (for `https://different.example.net/path/user`):

```
3045 <XRDS ref="https://different.example.net/path/user">
3046 <XRD>
3047 <EquivID>http://example.com/user</EquivID>
3048 <CanonicalID>https://different.example.net/path/user</CanonicalID>
3049 <Service priority="10">
3050 ...
3051 </Service>
3052 ...
3053 </XRD>
3054 </XRDS>
```

3055 The CanonicalEquivID asserted in the first XRDS satisfies the verification rules in section 14.3.3
3056 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of
3057 the first XRDS.

3058

3059 **Example #4:**

- 3060 • Fully-Qualified Query Identifier: `http://example.com/user`
- 3061 • Asserted CanonicalID: `http://example.com/user`
- 3062 • Asserted CanonicalEquivID: `!1000.62b1.44fd.2855`

3063 XRDS (for `http://example.com/user`):

```
3064 <XRDS ref="http://example.com/user">
3065 <XRD>
3066 <CanonicalID>http://example.com/user</CanonicalID>
3067 <CanonicalEquivID>xri://!1000.62b1.44fd.2855</CanonicalEquivID>
3068 <Service priority="10">
3069 ...
3070 </Service>
3071 ...
3072 </XRD>
3073 </XRDS>
```

3074 XRDS (for xri://=!1000.62b1.44fd.2855):

```
3075 <XRDS ref="xri://=!1000.62b1.44fd.2855">
3076 <XRD>
3077 <Query>!1000.62b1.44fd.2855</Query>
3078 <ProviderID>xri://=</ProviderID>
3079 <EquivID>http://example.com/user</EquivID>
3080 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3081 <Service priority="10">
3082 ...
3083 </Service>
3084 ...
3085 </XRD>
3086 </XRDS>
```

3087 The CanonicalEquivID asserted in the first XRDS satisfies the verification rules in section 14.3.3
3088 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of
3089 the first XRDS.

3090

3091 **Example #5:**

- 3092 • Fully-Qualified Query Identifier: =example.name
- 3093 • Asserted CanonicalID: xri://=!1000.62b1.44fd.2855
- 3094 • Asserted CanonicalEquivID: https://example.com/user

3095 First XRDS (for =example.name):

```
3096 <XRDS ref="xri://=example.name">
3097 <XRD>
3098 <Query>*example.name</Query>
3099 <ProviderID>xri://=</ProviderID>
3100 <LocalID>!1000.62b1.44fd.2855</LocalID>
3101 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3102 <CanonicalEquivID>https://example.com/user</CanonicalEquivID>
3103 <Service priority="10">
3104 ...
3105 </Service>
3106 ...
3107 </XRD>
3108 </XRDS>
```

3109 Second XRDS (for https://example.com/user):

```
3110 <XRDS ref="https://example.com/user">
3111 <XRD>
3112 <EquivID>xri://=!1000.62b1.44fd.2855</EquivID>
3113 <CanonicalID>https://example.com/user</CanonicalID>
3114 <Service priority="10">
3115 ...
3116 </Service>
3117 ...
3118 </XRD>
3119 </XRDS>
```

3120 The CanonicalEquivID asserted in the first XRDS satisfies the verification rules in section 14.3.3
3121 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of
3122 the first XRDS.

3123

3124 **Example #6:**

- 3125 • Fully-Qualified Query Identifier: =example.name*delegate.name
- 3126 • Asserted CanonicalID: xri://=!1000.62b1.44fd.2855!1234
- 3127 • Asserted CanonicalEquivID: @!1000.f3da.9056.aca3!5555

3128 First XRDS (for =example.name*delegate.name):

```
3129 <XRDS ref="xri://=example.name*delegate.name">
3130 <XRD>
3131   <Query>*example.name</Query>
3132   <ProviderID>xri://=example.name*delegate.name</ProviderID>
3133   <LocalID>!1000.62b1.44fd.2855</LocalID>
3134   <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3135   <Service>
3136     <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
3137     <Type>xri://$res*auth*($v*2.0)</Type>
3138     <MediaType>application/xrds+xml</MediaType>
3139     <URI priority="10">http://resolve.example.com</URI>
3140     <URI priority="15">http://resolve2.example.com</URI>
3141     <URI>https://resolve.example.com</URI>
3142   </Service>
3143   ...
3144 </XRD>
3145 <XRD>
3146   <Query>*delegate.name</Query>
3147   <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
3148   <LocalID>!1234</LocalID>
3149   <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
3150   <CanonicalEquivID>
3151     xri://@1000.f3da.9056.aca3!5555
3152   </CanonicalEquivID>
3153   <Service priority="1">
3154     ...
3155   </Service>
3156   ...
3157 </XRD>
3158 </XRDS>
```

3159 • Second XRDS (for @!1000.f3da.9056.aca3!5555):

```
3160 <XRDS ref="xri://@!1000.f3da.9056.aca3!5555">
3161 <XRD>
3162   <Query>!1000.f3da.9056.aca3</Query>
3163   <ProviderID>xri://@!1000.f3da.9056.aca3!5555</ProviderID>
3164   <CanonicalID>xri://@!1000.f3da.9056.aca3!5555</CanonicalID>
3165   <Service>
3166     <ProviderID>xri://@!1000.f3da.9056.aca3!5555</ProviderID>
3167     <Type>xri://$res*auth*($v*2.0)</Type>
3168     <MediaType>application/xrds+xml</MediaType>
3169     <URI priority="10">http://resolve.example.com</URI>
3170     <URI priority="15">http://resolve2.example.com</URI>
3171     <URI>https://resolve.example.com</URI>
3172   </Service>
3173   ...
3174 </XRD>
3175 <XRD>
3176   <Query>!5555</Query>
3177   <ProviderID>xri://@!1000.f3da.9056.aca3!5555</ProviderID>
3178   <LocalID>!5555</LocalID>
3179   <EquivID>xri://=!1000.62b1.44fd.2855!1234</EquivID>
```



```
3180 <CanonicalID>xri://@!1000.f3da.9056.aca3!5555</CanonicalID>
3181 <Service priority="1">
3182   ...
3183 </Service>
3184   ...
3185 </XRD>
3186 </XRDS>
```

3187 The CanonicalEquiVID asserted in the final XRD of the first XRDS satisfies the verification rules
3188 in section 14.3.3 because it resolves to a second XRDS whose final XRD asserts an EquiVID
3189 backpointer to the CanonicalID of the final XRD in the first XRDS.

3190

15 Status Codes and Error Processing

3191

15.1 Status Elements

3192

XRDS architecture uses two XRD elements for status reporting:

3193

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

3194

3195

- 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 14.3.4).

3196

3197

3198

3199

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

3200

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

3201

2. An XRI authority server is REQUIRED to include an `xrd:XRD/xrd:ServerStatus` element for each XRD in a resolution response.

3202

3203

BACKWARDS COMPATIBILITY 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.

3204

3205

3206

3207

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

3208

3209

4. In SAML trusted resolution, a resolver MUST verify the SAML signature on the XRD received from the server as specified in section 10.2.4 before adding the `xrd:XRD/xrd:Status` element to the XRD. Because this modifies the XRD, a consuming application may not be able to easily verify the SAML signature itself. Should this be necessary, the consuming application may request the XRD it wishes to verify directly from an authority server using the SAML trusted resolution protocol in section 10.2.

3210

3211

3212

3213

3214

3215

3216

5. These elements MUST include the status codes specified in section 15.2 as the value of the required `code` attribute.

3217

3218

6. These elements SHOULD contain the status context strings specified in section 15.3. Authority servers or resolvers MAY add additional information to status context strings.

3219

3220

15.2 Status Codes

3221

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

3222

3223

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

3224

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

3225

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

3226

3227 The 2xx and 3xx categories are broken into seven minor categories:

3228 • x0x: General error that may take place during any phase of resolution.

3229 • x1x: Input error

3230 • x2x: Generic authority resolution error.

3231 • x3x: Trusted authority resolution error.

3232 • x4x: Service endpoint (SEP) selection error.

3233 • x5x: Redirect error.

3234 • x6x: Ref error.

3235 The full list of XRI resolution status codes is defined in Table 29.

3236

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 implemented by the resolver.
202	LIMIT_EXCEEDED	Any	A locally configured resource limit was exceeded. Examples: number of Redirect or Refs 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 endpoint in an XRD.
222	QUERY_NOT_FOUND	Authority resolution	Responding authority does not have an XRI matching the query.
223	UNEXPECTED_XRD	Authority resolution	Value of the <code>xrd:Query</code> element does not match the subsegment requested.
224	INACTIVE	Authority resolution	The query XRI has been assigned but the authority does not provide resolution metadata.

230	TRUSTED_RES_ERROR	Trusted resolution	Generic trusted resolution error.
231	HTTPS_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS authority resolution endpoint.
232	SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate a SAML authority resolution endpoint.
233	HTTPS+SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS+SAML authority resolution endpoint.
234	UNVERIFIED_SIGNATURE	Trusted resolution	Signature verification failed.
240	SEP_SELECTION_ERROR	SEP selection	Generic service endpoint selection error.
241	SEP_NOT_FOUND	SEP selection	The requested service endpoint could not be found in the current XRD or via Redirect or Ref 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	<code>https=true</code> 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 12.3
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 <code>refs</code> parameter was set to <code>false</code> .
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 server, the server returned a non-200 HTTP status.
322	INVALID_XRDS	Authority resolution	Invalid XRDS received from an authority server (includes malformed XML, truncated content, or wrong content type).

3238 Table 29: Error codes for XRI resolution.

3239 **15.3 Status Context Strings**

3240 Each status code in Table 29 MAY be returned with an optional status context string that provides
3241 additional human-readable information about the status or error condition. When the Resolution
3242 Output Format is an XRDS document or XRD element, this string is returned as the contents of
3243 the `xrd:XRD/xrd:ServerStatus` and `xrd:XRD/xrd:Status` elements. When the
3244 Resolution Output Format is a URI List, this string MUST be returned as specified in section 15.4.
3245 Implementers SHOULD provide error context strings with additional information about an error
3246 and possible solutions whenever it can be helpful to developers or end users.

3247 **15.4 Returning Errors in Plain Text or HTML**

3248 If the Resolution Output Format is a URI List as defined in section 8.2, an error MUST be
3249 returned with the content type `text/plain`. In this content:

- 3250 • The first line MUST consist of only the numeric error code as defined in section 15.2 followed
3251 by a CRLF.
- 3252 • The second line is RECOMMENDED; if present it MUST contain the error context string as
3253 defined in section 15.3.

3254 The same rules apply if the Resolution Output Format is an HTTP(S) Redirect as defined in
3255 section 8.2, except the media type MAY also be `text/html`. It is particularly important in this
3256 case to return an error message that will be understandable to an end-user who may have no
3257 knowledge of XRI resolution or the fact that the error is coming from an XRI proxy resolver.

3258 **15.5 Error Handling in Recursing and Proxy Resolution**

3259 In recursing and proxy resolution (sections 9.1.8 and 11), a server is acting as a client resolver for
3260 other authority resolution service endpoints. If in this intermediary capacity it receives an
3261 unrecoverable error, it MUST return the error to the originating client in the output format
3262 specified by the value of the requested Resolution Output Format as defined in section 8.2.

3263 If the output format is an XRDS document, it MUST contain `xrd:XRD` elements for all
3264 subsegments successfully resolved or retrieved from cache prior to the error. Each XRD MUST
3265 include the `xrd:ServerStatus` element as reported by the authoritative server. The final
3266 `xrd:XRD` element MUST include the `xrd:Query` element that produced the error and the
3267 `xrd:Status` element that describes the error as defined above.

3268 If the output format is an XRD element, it MUST include the `xrd:Query` element that produced
3269 the error, the `xrd:ServerStatus` element as reported by the authoritative server, and the
3270 `xrd:Status` element that describes the error as defined above.

3271 If this output format is a URI List or an HTTP(S) redirect, a proxy resolver SHOULD return a
3272 human-readable error message as specified in section 15.4.

3273 16 Use of HTTP(S)

3274 16.1 HTTP Errors

3275 When a resolver encounters fatal HTTP(S) errors during the resolution process, it **MUST** return
3276 the appropriate XRI resolution error code and error message as defined in section 15. In this way
3277 calling applications do not have to deal separately with XRI and HTTP error messages.

3278 16.2 HTTP Headers

3279 16.2.1 Caching

3280 The HTTP caching capabilities described by **[RFC2616]** should be leveraged for all XRDS and
3281 XRI resolution protocols. Specifically, implementations **SHOULD** implement the caching model
3282 described in section 13 of **[RFC2616]**, and in particular, the “Expiration Model” of section 13.2, as
3283 this requires the fewest round-trip network connections.

3284 All XRI resolution servers **SHOULD** send the Cache-Control or Expires headers in their
3285 responses per section 13.2 of **[RFC2616]** unless there are overriding security or policy reasons to
3286 omit them.

3287 Note that HTTP Cache headers **SHOULD NOT** conflict with expiration information in an XRD.
3288 That is, the expiration date specified by HTTP caching headers **SHOULD NOT** be later than any
3289 of the expiration dates for any of the `xrd:Expires` elements returned in the HTTP response.
3290 This implies that recursing and proxy resolvers **SHOULD** compute the “soonest” expiration date
3291 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching
3292 headers for the HTTP response.

3293 16.2.2 Location

3294 During HTTP interaction, “Location” headers may be present per **[RFC2616]** (i.e., during 3XX
3295 redirects). Redirects **SHOULD** be made cacheable through appropriate HTTP headers, as
3296 specified in section 16.2.1.

3297 16.2.3 Content-Type

3298 For authority resolution, the Content-Type header in the 2XX responses **MUST** contain the media
3299 type identifier values specified in Table 11 (for generic resolution), Table 15 (for HTTPS trusted
3300 resolution), Table 16 (for SAML trusted resolution), or Table 17 (or HTTPS+SAML trusted
3301 resolution).

3302 Following the optional service endpoint selection phase, clients and servers **MAY** negotiate
3303 content type using standard HTTP content negotiation features. Regardless of whether this
3304 feature is used, however, the server **MUST** respond with an appropriate media type in the
3305 Content-Type header if the resource is found and an appropriate content type is returned.

3306 16.3 Other HTTP Features

3307 HTTP provides a number of other features including transfer-coding, proxying, validation-model
3308 caching, and so forth. All these features may be used insofar as they do not conflict with the
3309 required uses of HTTP described in this document.

3310 16.4 Caching and Efficiency

3311 16.4.1 Resolver Caching

3312 In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the
3313 application level. For best results, however, resolution clients SHOULD be conservative with
3314 caching expiration semantics, including cache expiration dates. This implies that in a series of
3315 HTTP redirects, for example, the results of the entire process SHOULD only be cached as long
3316 as the shortest period of time allowed by any of the intermediate HTTP responses.

3317 Because not all HTTP client libraries expose caching expiration to applications, identifier
3318 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the
3319 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments
3320 should be mindful of limitations in current HTTP clients and proxies.

3321 The cache expiration time of an XRD may also be explicitly limited by the parent authority. If the
3322 expiration time in the `xrd:Expires` element is sooner than the expiration time calculated from
3323 the HTTP caching semantics, the XRD MUST be discarded before the expiration time in
3324 `xrd:Expires`. Note also that a `saml:Assertion` element returned during SAML trusted
3325 resolution has its own signature expiration semantics as defined in [SAML]. While this may
3326 invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if
3327 it is not expired by HTTP caching semantics or the `xrd:Expires` element.

3328 With both application-level and HTTP-level caching, the resolution process is designed to have
3329 minimal overhead. Resolution of each qualified subsegment of an XRI authority component is a
3330 separate step described by a separate XRD, so intermediate results can typically be cached in
3331 their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified
3332 subsegments, which are common to more identifiers, will naturally result in a greater number of
3333 cache hits than resolution of lower-level subsegments.

3334 16.4.2 Synonyms

3335 The publication of synonyms in XRDS documents (section 5) can further increase cache
3336 efficiency. If an XRI resolution request produces a cache hit on a synonym, the following rules
3337 apply:

- 3338 1. If the cache hit is on a LocalID synonym, the resolver MAY return the cached XRD
3339 element if: a) it is from the correct ProviderID, b) it has not expired, and c) it was obtained
3340 using the same trusted resolution and synonym verification parameters as the current
3341 resolution request.
- 3342 2. If the cache hit is on a CanonicalID synonym, the resolver MAY return the entire cached
3343 XRDS document if: a) it has not expired, and b) it was obtained using the same trusted
3344 resolution and synonym verification parameters as the current resolution request.

3345 **IMPORTANT:** The effect of these rules is that the application calling an XRI resolver MAY receive
3346 back an XRD element, or an XRDS document containing XRD element(s), in which the value of
3347 the `<xrd:Query>` element does not match the resolution request, but in which the value of an
3348 `<xrd:LocalID>` element does match the resolution request. This is acceptable for the generic
3349 and HTTPS trusted resolution protocols but not the SAML trusted resolution protocol, where the
3350 value of the `<xrd:Query>` element MUST match the resolution request as specified in section
3351 10.2.4.

3352 17 Extensibility and Versioning

3353 17.1 Extensibility

3354 17.1.1 Extensibility of XRDs

3355 The XRD schema in Appendix B use an an open-content model that is designed to be extended
3356 with other metadata. In most places, extension elements and attributes from namespaces other
3357 than `xri://$xrd*($v*2.0)` are explicitly allowed. These extension points are designed to
3358 simplify default processing using a “Must Ignore” rule. The base rule is that unrecognized
3359 elements and attributes, and the content and child elements of unrecognized elements, MUST be
3360 ignored. As a consequence, elements that would normally be recognized by a processor MUST
3361 be ignored if they appear as descendants of an unrecognized element.

3362 Extension elements MUST NOT require new interpretation of elements defined in this document.
3363 If an extension element is present, a processor MUST be able to ignore it and still correctly
3364 process the XRDS document.

3365 Extension specifications MAY simulate “Must Understand” behavior by applying an “enclosure”
3366 pattern. Elements defined by the XRD schema in Appendix B whose meaning or interpretation is
3367 modified by extension elements can be wrapped in an extension container element defined by the
3368 extension specification. This extension container element SHOULD be in the same namespace
3369 as the other extension elements defined by the extension specification.

3370 Using this design, all elements whose interpretations are modified by the extension will now be
3371 contained in the extension container element and thus will be ignored by clients or other
3372 applications unable to process the extension. The following example illustrates this pattern using
3373 an extension container element from an extension namespace (`other:SuperService`) that
3374 contains an extension element (`other:ExtensionElement`):

```
3375 <XRD>  
3376   <Service>  
3377     ...  
3378   </Service>  
3379   <other:SuperService>  
3380     <Service>  
3381       ...  
3382       <other:ExtensionElement>...</other:ExtensionElement>  
3383     </Service>  
3384   </other:SuperService>  
3385 </XRD>
```

3386 In this example, the `other:ExtensionElement` modifies the interpretation or processing rules
3387 for the parent `xrd:Service` element and therefore must be understood by the consumer for the
3388 proper interpretation of the parent `xrd:Service` element. To preserve the correct interpretation
3389 of the `xrd:Service` element in this context, the `xrd:Service` element is “wrapped” in the
3390 `other:SuperService` element so only consumers that understand elements in the
3391 `other:SuperService` namespace will attempt to process the `xrd:Service` element.

3392 The addition of extension elements does not change the requirement for SAML signatures to be
3393 verified across all elements, whether recognized or not.

3394 Specifications extending XRDs MAY use the `xrd:XRD/xrd:Type` element to indicate to an XRD
3395 processor that an XRD conforms to the requirements of the extension specification. Such
3396 specification SHOULD be dereferenceable from the URI, IRI, or XRI used as the value of the

3397 `xrd:XRD/xrd:Type` element. However XRD processors MAY ignore instances of this element
3398 and process the XRD as specified in this document.

Comment [DSR22]: Added to support addition of `xrd:XRD/xrd:Type` element to 4.2.1.

3399 17.1.2 Other Points of Extensibility

3400 The use of HTTP(S), XML, XRIs, and URIs in the design of XRDS documents, XRD elements,
3401 and XRI resolution architecture provides additional specific points of extensibility:

- 3402 • Specification of new resolution service types or other service types using XRIs, IRIs, or URIs
3403 as values of the `xrd:Type` element.
- 3404 • Specification of new resolution output formats or features using media types and media type
3405 parameters as values of the `xrd:MediaType` element as defined in [RFC2045] and
3406 [RFC2046].
- 3407 • HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- 3408 • Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- 3409 • Use of cross-references within XRIs, particularly for associating new types of metadata with a
3410 resource. See [XRISyntax] and [XRIMetadata].

3411 17.2 Versioning

3412 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,
3413 this section describes versioning guidelines.

3414 In general, this specification follows the same versioning guidelines as established in section
3415 4.2.1 of [SAML]:

3416 *In general, maintaining namespace stability while adding or changing the content of a*
3417 *schema are competing goals. While certain design strategies can facilitate such changes,*
3418 *it is complex to predict how older implementations will react to any given change, making*
3419 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*
3420 *minor revisions is reserved, in the interest of namespace stability. Except in special*
3421 *circumstances (for example, to correct major deficiencies or to fix errors),*
3422 *implementations should expect forward-compatible schema changes in minor revisions,*
3423 *allowing new messages to validate against older schemas.*

3424 *Implementations SHOULD expect and be prepared to deal with new extensions and*
3425 *message types in accordance with the processing rules laid out for those types. Minor*
3426 *revisions MAY introduce new types that leverage the extension facilities described in [this*
3427 *section]. Older implementations SHOULD reject such extensions gracefully when they*
3428 *are encountered in contexts that dictate mandatory semantics.*

3429 17.2.1 Version Numbering

3430 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number
3431 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version
3432 number MajorA.MinorA if and only if:

3433 $Major_B > Major_A$ OR (($Major_B = Major_A$) AND $Minor_B > Minor_A$)

3434 17.2.2 Versioning of the XRI Resolution Specification

3435 New releases of the XRI Resolution specification may specify changes to the resolution protocols
3436 and/or the XRD schema in Appendix B. When changes affect either of these, the resolution
3437 service type version number will be changed. Where changes are purely editorial, the version
3438 number will not be changed.

3439 In general, if a change is backward-compatible, the new version will be identified using the
3440 current major version number and a new minor version number. If the change is not backward-
3441 compatible, the new version will be identified with a new major version number.

3442 **17.2.3 Versioning of Protocols**

3443 The protocols defined in this document may also be versioned by future releases of the XRI
3444 Resolution specification. If these protocols are not backward-compatible with older
3445 implementations, they will be assigned a new XRI with a new version identifier for use in
3446 identifying their service type in XRDs. See section 3.1.2.

3447 Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP
3448 provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an
3449 XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely
3450 to continue to use the same XRI to identify the protocol as was used in previous versions of the
3451 XRI Resolution specification.

3452 **17.2.4 Versioning of XRDs**

3453 The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have
3454 no specific knowledge of the elements it may contain. Therefore it has no version indicator, and
3455 can remain stable indefinitely because there is no need to version its namespace.

3456 The `xrd:XRD` element has a `version` attribute. This attribute is OPTIONAL for this version of
3457 the XRI resolution specification (version 2.0). This attribute will be REQUIRED for all future
3458 versions of this specification. When used, the value of this attribute MUST be the exact numeric
3459 version value of the XRI Resolution specification to which its containing elements conform.

3460 When new versions of the XRI Resolution specification are released, the namespace for the XRD
3461 schema may or may not be changed. If there is a major version number change, the namespace
3462 for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the
3463 namespace for the `xrd:XRD` schema may remain unchanged.

3464 Note that conformance to a specific XRD version does not preclude an author from including
3465 extension elements from a different namespace in the XRD. See section 17.1 above.

3466

18 Security and Data Protection

3467
3468
3469
3470

Significant portions of this specification deal directly with security issues; 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.

3471

18.1 DNS Spoofing or Poisoning

3472
3473
3474
3475
3476
3477
3478
3479
3480

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.

3481

18.2 HTTP Security

3482
3483
3484
3485

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.

3486
3487
3488
3489

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.

3490
3491

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

3492

18.3 SAML Considerations

3493
3494
3495

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.

3496

18.4 Limitations of Trusted Resolution

3497
3498
3499
3500
3501

While the trusted resolution protocols specified in this document provide 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.

3502
3503
3504

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.

3505 **18.5 Synonym Verification**

3506 As discussed in section 5, XRI and XRDS infrastructure has rich support for identifier synonyms,
3507 including synonyms that cross security domains. For this reason it is particularly important that
3508 identifier authorities, including registries, registrars, directory administrators, identity providers,
3509 and other parties who issue XRIs and manage XRDS documents, enforce the security policies
3510 highlighted in section 5 regarding registration and management of XRDS synonym elements.

3511 **18.6 Redirect and Ref Management**

3512 As discussed in sections 5.3 and 12, XRI and XRDS infrastructure includes the capability to
3513 distribute and delegate XRDS document management across multiple network locations or
3514 identifier authorities. Identifier authorities should follow the security precautions highlighted in
3515 section 5.3 to ensure Redirects and Refs are properly authorized and represent the intended
3516 delegation policies.

3517 **18.7 Community Root Authorities**

3518 The XRI authority information for a community root needs to be well-known to the clients that
3519 request resolution within that community. For trusted resolution, this includes the authority
3520 resolution service endpoint URIs, the `xrd:XRD/xrd:ProviderID`, and the `ds:KeyInfo`
3521 information. An acceptable means of providing this information is for the community root authority
3522 to produce a self-signed XRD and publish it to a server-authenticated HTTPS endpoint. Special
3523 care should be taken to ensure the correctness of such an XRD; if this information is incorrect, an
3524 attacker may be able to convince a client of an incorrect result during trusted resolution.

3525 **18.8 Caching Authorities**

3526 In addition to traditional HTTP caching proxies, XRI proxy resolvers may be a part of the
3527 resolution topology. Such proxy resolvers should take special precautions against cache
3528 poisoning, as these caching entities may represent trusted decision points within a deployment's
3529 resolution architecture.

3530 **18.9 Recursing and Proxy Resolution**

3531 During recursing resolution, subsegments of the XRI authority component for which the resolving
3532 network endpoint is not authoritative may be revealed to that service endpoint. During proxy
3533 resolution, some or all of an XRI is provided to the proxy resolver.

3534 In both cases, privacy considerations should be evaluated before disclosing such information.

3535 **18.10 Denial-Of-Service Attacks**

3536 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks
3537 typical of systems relying on DNS and HTTP(S).

3538

A. Acknowledgments

3539 The editors would like to thank the following current and former members of the OASIS XRI TC
3540 for their particular contributions to this and previous versions of this specification:

- 3541 • William Barnhill, Booz Allen and Hamilton
- 3542 • Dave McAlpin, Epok
- 3543 • Chetan Sabnis, Epok
- 3544 • Peter Davis, Neustar
- 3545 • Victor Grey, PlaNetwork
- 3546 • Mike Lindelsee, Visa International
- 3547 • Markus Sabadello, XDI.org
- 3548 • John Bradley
- 3549 • Kermit Snelson

3550 The editors would also like to acknowledge the contributions of the other members of the OASIS
3551 XRI Technical Committee, whose other voting members at the time of publication were:

- 3552 • Geoffrey Strongin, Advanced Micro Devices
- 3553 • Ajay Madhok, AmSoft Systems
- 3554 • Dr. XiaoDong Lee, China Internet Network Information
- 3555 • Nat Sakimura, Nomura Research
- 3556 • Owen Davis, PlaNetwork
- 3557 • Fen Labalme, PlaNetwork
- 3558 • Marty Schleiff, The Boeing Company
- 3559 • Dave Wentker, Visa International
- 3560 • Paul Trevithick

3561 The editors also would like to acknowledge the following people for their contributions to previous
3562 versions of OASIS XRI specifications (affiliations listed for OASIS members):

3563 Marc Le Maitre, Cordance Corporation; Thomas Bikeev, EAN International; Krishna Sankar,
3564 Cisco; Winston Bumpus, Dell; Joseph Moeller, EDS; Steve Green, Epok; Lance Hood, Jerry
3565 Kindall, Adarbad Master, Davis McPherson, Chetan Sabnis, and Loren West, Epok; Phillipe
3566 LeBlanc, Jim Schreckengast, and Xavier Serret, Gemplus; John McGarvey, IBM; Reva Modi,
3567 Infosys; Krishnan Rajagopalan, Novell; Masaki Nishitani, Tomonori Seki, and Tetsu Watanabe,
3568 Nomura Research Institute; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,
3569 Verisign; Rajeev Maria, Terence Spielman, and John Veizades, Visa International; Lark Allen and
3570 Michael Willett, Wave Systems; Matthew Dovey; Eamonn Neylon; Mary Nishikawa; Lars Marius
3571 Garshol; Norman Paskin; and Bernard Vatant.

3572

B. RelaxNG Schema for XRDS and XRD

3573 Following are the locations of the normative RelaxNG compact schema files for XRDS and XRD
3574 as defined by this specification:

- 3575 • **xrds.rnc**: <http://docs.oasis-open.org/xri/2.0/specs/cd02/xrds.rnc>
- 3576 • **xrd.rnc**: <http://docs.oasis-open.org/xri/2.0/specs/cd02/xrd.rnc>

3577 **IMPORTANT:** The **xrd.rnc** schema does NOT include deprecated attribute values that are
3578 recommended for backwards compatibility. See the highlighted Backwards Compatibility notes in
3579 sections 9.1.1 and 13.3.2 for more details.

3580 Listings of these files are provided in this appendix for reference but are non-normative.

3581 **xrds.rnc**

```
3582 namespace xrds = "xri://$xrds"  
3583 namespace xrd = "xri://$xrd*($v*2.0)"  
3584 namespace local = ""  
3585 datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"  
3586  
3587 any.element =  
3588   element * {  
3589     (attribute * {text} *  
3590     | text  
3591     | any.element)*  
3592   }  
3593  
3594 any.external.element =  
3595   element * - (xrd:XRD | xrds:XRDS) {  
3596     (attribute * {text} *  
3597     | text  
3598     | any.element)*  
3599   }  
3600  
3601 other.attribute = attribute * - (local:*) {text}  
3602  
3603 start = XRDS  
3604  
3605 XRDS = element xrds:XRDS {  
3606   other.attribute *,  
3607   (attribute ref {xs:anyURI} | attribute redirect {xs:anyURI} )? ,  
3608   (any.external.element | XRDS | external "xrd.rnc")*  
3609 }  
3610
```

3611 **xrd.rnc**

```
3612 default namespace = "xri://$xrd*($v*2.0)"  
3613 namespace xrd = "xri://$xrd*($v*2.0)"  
3614 namespace saml = "urn:oasis:names:tc:SAML:2.0:assertion"  
3615 namespace ds = "http://www.w3.org/2000/09/xmldsig#"  
3616 namespace local = ""  
3617  
3618 datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"  
3619  
3620 start = XRD  
3621  
3622 anyelementbody =  
3623   (attribute * {text}  
3624   | text  
3625   | element * {anyelementbody} )*  
3626
```

```

3627 non.xrd.element = element * - xrd:* {
3628     anyelementbody
3629 }
3630
3631 other.attribute = attribute * - (local:* | xrd:* ) {text}
3632
3633
3634 XRD = element XRD {
3635     other.attribute *,
3636     attribute idref {xs:IDREF} ?,
3637     attribute version {"2.0"} ?,
3638     Type *,
3639     Query ?,
3640     Status ?,
3641     ServerStatus ?,
3642     Expires ?,
3643     ProviderID ?,
3644     (Redirect+ | Ref+) ?,
3645     LocalID *,
3646     EquipID *,
3647     CanonicalID ?,
3648     CanonicalEquipID ?,
3649     Service *,
3650     element saml:Assertion {anyelementbody} ?,
3651     non.xrd.element *
3652 }
3653
3654 Query = element Query {
3655     other.attribute *,
3656     text
3657 }
3658
3659 statuspattern =
3660     other.attribute *,
3661     attribute code {xs:integer},
3662     attribute cid {"absent" | "off" | "verified" | "failed"} ?,
3663     attribute ceid {"absent" | "off" | "verified" | "failed"} ?,
3664     text
3665
3666 Status = element Status {
3667     statuspattern
3668 }
3669
3670 ServerStatus = element ServerStatus {
3671     statuspattern
3672 }
3673
3674 Expires = element Expires {
3675     other.attribute *,
3676     xs:dateTime
3677 }
3678
3679 ProviderID = element ProviderID {
3680     other.attribute *,
3681     xs:anyURI
3682 }
3683
3684 Redirect = element Redirect {
3685     other.attribute *,
3686     attribute priority {xs:integer}?,
3687     xs:anyURI
3688 }
3689
3690 Ref = element Ref{
3691     other.attribute *,
3692     attribute priority {xs:integer}?,
3693     xs:anyURI
3694 }
3695

```

Comment [DSR23]: Added per new xrd:XRD/xrd:Type element in section 4.2.1. Needs proofing by Gabe.

Comment [DSR24]: The + was missing on this elements.


```

3696 LocalID = element LocalID {
3697     other.attribute *,
3698     attribute priority {xs:integer} ?,
3699     xs:anyURI
3700 }
3701
3702 EquivID = element EquivID {
3703     other.attribute *,
3704     attribute priority {xs:integer} ?,
3705     xs:anyURI
3706 }
3707
3708 CanonicalID = element CanonicalID {
3709     other.attribute *,
3710     xs:anyURI
3711 }
3712
3713 CanonicalEquivID = element CanonicalEquivID {
3714     other.attribute *,
3715     xs:anyURI
3716 }
3717
3718 Service = element Service {
3719     other.attribute *,
3720     attribute priority {xs:integer}?,
3721     ProviderID?,
3722     Type *,
3723     Path *,
3724     MediaType *,
3725     (URI+ | Redirect+ | Ref+)?,
3726     LocalID *,
3727     element ds:KeyInfo {anyelementbody}?,
3728     non.xrd.element *
3729 }
3730
3731 URI = element URI {
3732     other.attribute *,
3733     attribute priority {xs:integer}?,
3734     attribute append {"none" | "local" | "authority" | "path" | "query" | "qxri"} ?,
3735     xs:anyURI
3736 }
3737
3738 selection.attributes = attribute match {"any" | "default" | "non-null" | "null"} ?,
3739     attribute select {xs:boolean} ?
3740
3741 Type = element Type {
3742     other.attribute *,
3743     selection.attributes,
3744     xs:anyURI
3745 }
3746
3747 Path = element Path {
3748     other.attribute *,
3749     selection.attributes,
3750     xs:string
3751 }
3752
3753 MediaType = element MediaType {
3754     other.attribute *,
3755     selection.attributes,
3756     xs:string
3757 }

```

3758

C. XML Schema for XRDS and XRD

3759 Following are the locations of the non-normative W3C XML Schema files for XRDS and XRD as
3760 defined by this specification. Note that these are provided for reference only as they are not able
3761 to fully express the extensibility semantics of the RelaxNG versions.

3762 • **xrds.xsd**: <http://docs.oasis-open.org/xri/2.0/specs/cd02/xrds.xsd>

3763 • **xrd.xsd**: <http://docs.oasis-open.org/xri/2.0/specs/cd02/xrd.xsd>

3764 **IMPORTANT**: The **xrd.xsd** schema does NOT include deprecated attribute values that are
3765 recommended for backwards compatibility. See the highlighted Backwards Compatibility notes in
3766 sections 9.1.1 and 13.3.2 for more details.

3767 Listings of these files are provided in this appendix for reference.

3768 **xrds.xsd**

```
3769 <?xml version="1.0" encoding="UTF-8"?>
3770 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds"
3771 targetNamespace="xri://$xrds" elementFormDefault="qualified">
3772   <!-- Utility patterns -->
3773   <xs:attributeGroup name="otherattribute">
3774     <xs:anyAttribute namespace="##other" processContents="lax"/>
3775   </xs:attributeGroup>
3776   <xs:group name="otherelement">
3777     <xs:choice>
3778       <xs:any namespace="##other" processContents="lax"/>
3779       <xs:any namespace="##local" processContents="lax"/>
3780     </xs:choice>
3781   </xs:group>
3782   <!-- Patterns for elements -->
3783   <xs:element name="XRDS">
3784     <xs:complexType>
3785       <xs:sequence>
3786         <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3787       </xs:sequence>
3788       <xs:attributeGroup ref="xrds:otherattribute"/>
3789       <!--XML Schema does not currently offer a means to express that only one of
3790 the following two attributes may be used in any XRDS element, i.e., an XRDS document may
3791 describe EITHER a redirect identifier or a ref identifier but not both.-->
3792       <xs:attribute name="redirect" type="xs:anyURI" use="optional"/>
3793       <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
3794     </xs:complexType>
3795   </xs:element>
3796 </xs:schema>
3797
3798
```

3799 **xrd.xsd**

```
3800 <?xml version="1.0" encoding="UTF-8"?>
3801 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3802 xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xrd="xri://$xrd*($v*2.0)"
3803 targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified">
3804   <!-- Utility patterns -->
3805   <xs:attributeGroup name="otherattribute">
3806     <xs:anyAttribute namespace="##other" processContents="lax"/>
3807   </xs:attributeGroup>
3808   <xs:group name="otherelement">
3809     <xs:choice>
3810       <xs:any namespace="##other" processContents="lax"/>
3811       <xs:any namespace="##local" processContents="lax"/>
3812     </xs:choice>
3813   </xs:group>
```

```

3814 <xs:attributeGroup name="priorityAttrGrp">
3815   <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
3816 </xs:attributeGroup>
3817 <xs:attributeGroup name="codeAttrGrp">
3818   <xs:attribute name="code" type="xs:int" use="required"/>
3819 </xs:attributeGroup>
3820 <xs:attributeGroup name="verifyAttrGrp">
3821   <xs:attribute name="cid" use="optional">
3822     <xs:simpleType>
3823       <xs:restriction base="xs:string">
3824         <xs:enumeration value="absent"/>
3825         <xs:enumeration value="off"/>
3826         <xs:enumeration value="verified"/>
3827         <xs:enumeration value="failed"/>
3828       </xs:restriction>
3829     </xs:simpleType>
3830   </xs:attribute>
3831   <xs:attribute name="ceid" use="optional">
3832     <xs:simpleType>
3833       <xs:restriction base="xs:string">
3834         <xs:enumeration value="absent"/>
3835         <xs:enumeration value="off"/>
3836         <xs:enumeration value="verified"/>
3837         <xs:enumeration value="failed"/>
3838       </xs:restriction>
3839     </xs:simpleType>
3840   </xs:attribute>
3841 </xs:attributeGroup>
3842 <xs:attributeGroup name="selectionAttrGrp">
3843   <xs:attribute name="match" use="optional" default="default">
3844     <xs:simpleType>
3845       <xs:restriction base="xs:string">
3846         <xs:enumeration value="default"/>
3847         <xs:enumeration value="any"/>
3848         <xs:enumeration value="non-null"/>
3849         <xs:enumeration value="null"/>
3850       </xs:restriction>
3851     </xs:simpleType>
3852   </xs:attribute>
3853   <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
3854 </xs:attributeGroup>
3855 <xs:attributeGroup name="appendAttrGrp">
3856   <xs:attribute name="append" use="optional" default="none">
3857     <xs:simpleType>
3858       <xs:restriction base="xs:string">
3859         <xs:enumeration value="none"/>
3860         <xs:enumeration value="local"/>
3861         <xs:enumeration value="authority"/>
3862         <xs:enumeration value="path"/>
3863         <xs:enumeration value="query"/>
3864         <xs:enumeration value="qxri"/>
3865       </xs:restriction>
3866     </xs:simpleType>
3867   </xs:attribute>
3868 </xs:attributeGroup>
3869 <xs:complexType name="URIPattern">
3870   <xs:simpleContent>
3871     <xs:extension base="xs:anyURI">
3872       <xs:attributeGroup ref="xrd:otherattribute"/>
3873     </xs:extension>
3874   </xs:simpleContent>
3875 </xs:complexType>
3876 <xs:complexType name="URIPriorityPattern">
3877   <xs:simpleContent>
3878     <xs:extension base="xrd:URIPattern">
3879       <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3880     </xs:extension>
3881   </xs:simpleContent>
3882 </xs:complexType>

```

```

3883 <xs:complexType name="URIPriorityAppendPattern">
3884 <xs:simpleContent>
3885 <xs:extension base="xrd:URIPriorityPattern">
3886 <xs:attributeGroup ref="xrd:appendAttrGrp" />
3887 </xs:extension>
3888 </xs:simpleContent>
3889 </xs:complexType>
3890 <xs:complexType name="StringPattern">
3891 <xs:simpleContent>
3892 <xs:extension base="xs:string">
3893 <xs:attributeGroup ref="xrd:otherattribute" />
3894 </xs:extension>
3895 </xs:simpleContent>
3896 </xs:complexType>
3897 <xs:complexType name="StringSelectionPattern">
3898 <xs:simpleContent>
3899 <xs:extension base="xrd:StringPattern">
3900 <xs:attributeGroup ref="xrd:selectionAttrGrp" />
3901 </xs:extension>
3902 </xs:simpleContent>
3903 </xs:complexType>
3904 <!-- Patterns for elements -->
3905 <xs:element name="XRD">
3906 <xs:complexType>
3907 <xs:sequence>
3908 <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded" />
3909 <xs:element ref="xrd:Query" minOccurs="0" />
3910 <xs:element ref="xrd:Status" minOccurs="0" />
3911 <xs:element ref="xrd:ServerStatus" minOccurs="0" />
3912 <xs:element ref="xrd:Expires" minOccurs="0" />
3913 <xs:element ref="xrd:ProviderID" minOccurs="0" />
3914 <xs:choice>
3915 <xs:element ref="xrd:Redirect" minOccurs="0" maxOccurs="unbounded" />
3916 <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded" />
3917 </xs:choice>
3918 <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded" />
3919 <xs:element ref="xrd:EquipID" minOccurs="0" maxOccurs="unbounded" />
3920 <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded" />
3921 <xs:element ref="xrd:CanonicalEquipID" minOccurs="0"
3922 maxOccurs="unbounded" />
3923 <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded" />
3924 <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded" />
3925 </xs:sequence>
3926 <xs:attribute name="idref" type="xs:IDREF" use="optional" />
3927 <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0" />
3928 <xs:attributeGroup ref="xrd:otherattribute" />
3929 </xs:complexType>
3930 </xs:element>
3931 <xs:element name="Query" type="xrd:StringPattern" />
3932 <xs:element name="Status">
3933 <xs:complexType>
3934 <xs:simpleContent>
3935 <xs:extension base="xrd:StringPattern">
3936 <xs:attributeGroup ref="xrd:codeAttrGrp" />
3937 <xs:attributeGroup ref="xrd:verifyAttrGrp" />
3938 <xs:attributeGroup ref="xrd:otherattribute" />
3939 </xs:extension>
3940 </xs:simpleContent>
3941 </xs:complexType>
3942 </xs:element>
3943 <xs:element name="ServerStatus">
3944 <xs:complexType>
3945 <xs:simpleContent>
3946 <xs:extension base="xrd:StringPattern">
3947 <xs:attributeGroup ref="xrd:codeAttrGrp" />
3948 <xs:attributeGroup ref="xrd:otherattribute" />
3949 </xs:extension>
3950 </xs:simpleContent>
3951 </xs:complexType>
3952 </xs:element>

```

Comment [DSR25]: New per 4.2.1. Gabe needs to proof. Also, if we reuse the Type element, should we move its order not within the XRD element, but within this schema document (i.e., put it before Query)?

```

3953 <xs:element name="Expires">
3954   <xs:complexType>
3955     <xs:simpleContent>
3956       <xs:extension base="xs:dateTime">
3957         <xs:attributeGroup ref="xrd:otherattribute"/>
3958       </xs:extension>
3959     </xs:simpleContent>
3960   </xs:complexType>
3961 </xs:element>
3962 <xs:element name="ProviderID" type="xrd:URIPattern"/>
3963 <xs:element name="Redirect" type="xrd:URIPriorityAppendPattern"/>
3964 <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
3965 <xs:element name="LocalID">
3966   <xs:complexType>
3967     <xs:simpleContent>
3968       <xs:extension base="xrd:StringPattern">
3969         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3970       </xs:extension>
3971     </xs:simpleContent>
3972   </xs:complexType>
3973 </xs:element>
3974 <xs:element name="EquivID" type="xrd:URIPriorityPattern"/>
3975 <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>
3976 <xs:element name="CanonicalEquivID" type="xrd:URIPriorityPattern"/>
3977 <xs:element name="Service">
3978   <xs:complexType>
3979     <xs:sequence>
3980       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
3981       <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
3982       <xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
3983       <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
3984       <xs:choice>
3985         <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
3986         <xs:element ref="xrd:Redirect" minOccurs="0" maxOccurs="unbounded"/>
3987         <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
3988       </xs:choice>
3989       <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
3990       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3991     </xs:sequence>
3992     <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3993     <xs:attributeGroup ref="xrd:otherattribute"/>
3994   </xs:complexType>
3995 </xs:element>
3996 <xs:element name="Type">
3997   <xs:complexType>
3998     <xs:simpleContent>
3999       <xs:extension base="xrd:URIPattern">
4000         <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
4001       </xs:extension>
4002     </xs:simpleContent>
4003   </xs:complexType>
4004 </xs:element>
4005 <xs:element name="Path" type="xrd:StringSelectionPattern"/>
4006 <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
4007 <xs:element name="URI" type="xrd:URIPriorityAppendPattern"/>
4008 </xs:schema>
4009

```

4010

D. Media Type Definition for application/xrds+xml

4011 This section is prepared in anticipation of filing a media type registration meeting the
4012 requirements of [RFC4288].

4013 **Type name:** application

4014 **Subtype name:** xrds+xml

4015 **Required parameters:** None

4016 **Optional parameters:** See Table 6 of this document.

4017 **Encoding considerations:** Identical to those of "application/xml" as described in [RFC3023],
4018 Section 3.2.

4019 **Security considerations:** As defined in this specification. In addition, as this media type uses the
4020 "+xml" convention, it shares the same security considerations as described in [RFC3023],
4021 Section 10.

4022 **Interoperability considerations:** There are no known interoperability issues.

4023 **Published specification:** This specification.

4024 **Applications that use this media type:** Applications conforming to this specification use this
4025 media type.

4026 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
4027 Technical Committee Co-Chair, drummond.reed@cordance.net

4028 **Intended usage:** COMMON

4029 **Restrictions on usage:** None

4030 **Author:** OASIS XRI TC

4031 **Change controller:** OASIS XRI TC

4032

E. Media Type Definition for application/xrd+xml

4033 This section is prepared in anticipation of filing a media type registration meeting the
4034 requirements of [RFC4288].

4035 **Type name:** application

4036 **Subtype name:** xrd+xml

4037 **Required parameters:** None

4038 **Optional parameters:** See Table 6 of this document.

4039 **Encoding considerations:** Identical to those of "application/xml" as described in [RFC3023],
4040 Section 3.2.

4041 **Security considerations:** As defined in this specification. In addition, as this media type uses the
4042 "+xml" convention, it shares the same security considerations as described in [RFC3023],
4043 Section 10.

4044 **Interoperability considerations:** There are no known interoperability issues.

4045 **Published specification:** This specification.

4046 **Applications that use this media type:** Applications conforming to this specification use this
4047 media type.

4048 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI
4049 Technical Committee Co-Chair, drummond.reed@cordance.net

4050 **Intended usage:** COMMON

4051 **Restrictions on usage:** None

4052 **Author:** OASIS XRI TC

4053 **Change controller:** OASIS XRI TC

4054

F. Example Local Resolver Interface Definition

4055

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

4056

4057

The interface definition is provided as five operations where each operation takes two or more of the following input parameters. These input parameters correspond to the normative text in section 8.1. In all of these parameters, the value empty string ("") is interpreted the same as the value null.

4058

4059

4060

4061

Parameter name	Description
QXRI	Query XRI as defined in section 8.1.1.
sepType	Service Types as defined in section 8.1.3
sepMediaType	Service Media Type as defined in section 8.1.4
flags	Language binding-specific representation of resolution flags defined in the following table.

4062

4063

The `flags` parameter is a binding-specific container data structure that encapsulates the following subparameters of the Resolution Output Format parameter. All of these are Boolean parameters defined in Table 6 in section 3.3.

4064

4065

4066

Subparameter	Description
https, saml	Specifies use of HTTPS or SAML trusted resolution as defined in sections 10.1 and 10.2.
refs	Specifies whether Refs should be followed during resolution as defined in section 12.4.
nodefault_t, nodefault_p, nodefault_m	Specifies whether a default match is allowed on the Type, Path, or MediaType elements respectively during service endpoint selection as defined in section 13.3.
uric	Specifies whether a resolver should automatically construct service endpoint URIs as defined in section 13.7.1.
cid	Specifies whether automatic canonical ID verification should performed as defined in section 14.3.

4067

4068

Note that one subparameter defined in in Table 6, `sep` (service endpoint), is not included in this flags table because it is implicitly represented in the operation being called. The five operations shown in the table below correspond to the five possible combinations of the value of the Resolution Output Format parameter and the `sep` subparameter. (Note that if the Resolution Output Format is URI List, the `sep` subparameter MUST be considered to be TRUE, so there is no `resolveAuthToURIList` operation.)

4069

4070

4071

4072

4073

4074

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

4075 Following is the API and descriptions of the five operations.

4076 **1. Resolve Authority to XRDS**

```
4077 Result resolveAuthToXRDS(  
4078     in string QXRI, in Flags flags);
```

- 4079 • Performs authority resolution only (sections 9 and 10) and outputs an XRDS document as
4080 specified in section 8.2.1 when the `sep` subparameter is FALSE.
- 4081 • Only the authority component of the QXRI is processed by this function. If the QXRI contains
4082 a path or query component, it is ignored.
- 4083 • Returns a binding-specific representation of the resolution result which may include, but is not
4084 limited to, XRDS output, success/failure code, exceptions and error context.
- 4085 • The XRD element(s) in the output XRDS will be signed or not depending on the value of the
4086 `saml` flag.

4087

4088 **2. Resolve Authority to XRD**

```
4089 Result resolveAuthToXRD(  
4090     in string QXRI, in Flags flags);
```

- 4091 • Performs authority resolution only (sections 9 and 10) and outputs an XRD element as
4092 specified in section 8.2.2 when the `sep` subparameter is FALSE.
- 4093 • Only the authority component of the QXRI is processed by this function. If the QXRI contains
4094 a path or query component, it is ignored.
- 4095 • Returns a binding-specific representation of the resolution result which may include, but is not
4096 limited to, XRD output, success/failure code, exceptions and error context.
- 4097 • The output XRD will be signed or not depending on the value of the `saml` flag.

4098

4099 3. Resolve Service Endpoint to XRDS

```
4100 Result resolveSEPToXRDS (  
4101     in string QXRI, in string sepType,  
4102     in string sepMediaType, in Flags flags);
```

- 4103 • Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13)
4104 and outputs the XRDS as specified in section 8.2.1 when the `sep` subparameter is TRUE.
- 4105 • Returns a binding-specific representation of the resolution result which may include, but is not
4106 limited to, XRDS output, success/failure code, exceptions and error context.
- 4107 • The final XRD in the output XRDS will either contain at least one instance of the requested
4108 service endpoint or an error. *IMPORTANT: Although the resolver will perform service
4109 selection, the final XRD is NOT filtered when the Resolution Output Format is an XRDS
4110 document. Filtering is only performed when the Resolution Output Format is an XRD
4111 document (below).*
- 4112 • The XRD element(s) in the output XRDS will be signed or not depending on the value of
4113 `saml` flag.

4114

4115 4. Resolve Service Endpoint to XRD

```
4116 Result resolveSEPToXRD (  
4117     in string QXRI, in string sepType,  
4118     in string sepMediaType, in Flags flags);
```

- 4119 • Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13)
4120 and outputs an XRD as specified in section 8.2.2 when the `sep` subparameter is TRUE.
- 4121 • Returns a binding-specific representation of the resolution result which may include, but is not
4122 limited to, XRD output, success/failure code, exceptions and error context.
- 4123 • The output XRD will contain at least one instance of the requested service endpoint or an
4124 error. Also, all elements in the output XRD subject to the global `priority` attribute will be
4125 returned in order of highest to lowest priority. See section 8.2.2 for details.
- 4126 • The XRD element will be signed or not depending on the value of `saml` flag, however that
4127 signature may not be able to be independently verified because the XRD has been filtered to
4128 contain only the selected service endpoints.

4129

4130 **5. Resolve Service Endpoint to URI List**

```
4131 Result resolveSepToURIList(  
4132     in string QXRI, in string sepType,  
4133     in string sepMediaType, in Flags flags);
```

- 4134
- 4135 • Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13) and outputs a non-empty URI List or an error as specified in section 8.2.3.
 - 4136 • Returns a binding-specific representation of the resolution result which may include, but not
4137 limited to, URI-list output, success/failure code, exceptions and error context.
 - 4138 • If successful, the output URI-list will contain zero or more elements. It is possible that the
4139 selected service contains no URI element and it is up to the consuming application to
4140 interpret such a result.
- 4141

4142 G. Revision History

4143 Committee Draft 01 of this specification was published in March 2005 and is available at:

- 4144 • <http://www.oasis-open.org/committees/download.php/11853>

4145 Significant changes were made based on implementation feedback, resulting in a new
4146 implementers draft (Working Draft 10) published in March 2006:

- 4147 • <http://www.oasis-open.org/committees/download.php/17293>

4148 All revisions since Working Draft 10 have been tracked on the XRI Technical Committee wiki
4149 page for Working Draft 11:

- 4150 • <http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11>

4151 A copy of this wiki page as of the date of this specification has been archived at:

- 4152 • <http://www.oasis-open.org/committees/download.php/26277>

4153 Due to the extent of the revisions from Committee Draft 01, Committee Draft 02 should be
4154 considered a new document.

4155 Committee Draft 03 includes the following revisions based on comments received during the
4156 public review of Committee Draft 02:

Comment [DSR26]: Added to track CD03 revisions.

- 4157 • The reference to the XRI Syntax 2.0 specification in section 1.5 was updated.
- 4158 • The XRD elements in sections 4.2.1 – 4.2.6 were reformatted to include attribute definitions
4159 as separate bullet points (per comment received from Eran Hammer-Lahav).
- 4160 • The `xrd:XRD/xrd:Type` element was added to the XRD schema (section 4.2.1 and
4161 Appendix B and C) to reuse the `xrd:XRD/xrd:Service/xrd:Type` element at the XRD
4162 level in order to support extension specifications (per comment received from Eran Hammer-
4163 Lahav).
- 4164 • The flowcharts in Figures 5, 6, 7, and 8 were edited for improved clarity about recording
4165 XRDs and nested XRDS documents and clarify using a Redirect URI as an input.
- 4166 • The Next Authority URI construction algorithm in section 9.1.10 was loosened slightly.
- 4167 • The wording of the bullet points in section 12.1 were clarified (per comment received from
4168 Eran Hammer-Lahav).
- 4169 • Added fourth example in section 12.5.1 to illustrate double XRDS nesting.
- 4170 • Added clarifications to pseudocode in section 13.6.
- 4171 • The CanonicalID verification rule for XRIs was simplified to eliminate the need to involve the
4172 `xrd:XRD/xrd:ProviderID` element (per suggestion from editor William Tan).

4173