

# Extensible Resource Identifier (XRI) Resolution Version 2.0

# **Committee Specification 01**

# 12 April 2008

#### **Specification URIs:**

#### This Version:

http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cs01/xri-resolution-V2.0-cs-01.html http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cs01/xri-resolution-V2.0-cs-01.pdf http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cs01/xri-resolution-V2.0-cs-01.doc (Authoritative)

#### **Previous Version:**

http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xri-resolution-V2.0-cd-03.html http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xri-resolution-V2.0-cd-03.pdf http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xri-resolution-V2.0-cd-03.doc (Authoritative)

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

#### Status:

This document was last revised or approved by the XRI Technical Committee on the above date. The level of approval is also listed above. Check the "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.oasisopen.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 <a href="http://www.oasis-open.org/who/trademark.php">http://www.oasis-open.org/who/trademark.php</a> 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	
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	
	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	1.11 Recursing Authority Resolution	57
9.2 IF	RI Authority Resolution	58
9.2	2.1 Service Type and Media Type	58
9.2	2.2 Protocol	58
9.2	2.3 Optional Use of HTTPS	58
10 Tri	usted Authority Resolution Service	60
10.1	HTTPS	60
10	0.1.1 Service Type and Service Media Type	60
10	).1.2 Protocol	60
10.2	SAML	60
10	0.2.1 Service Type and Service Media Type	61
10	0.2.2 Protocol	61
10	0.2.3 Recursing Authority Resolution	62
10	0.2.4 Client Validation of XRDs	63
10	0.2.5 Correlation of ProviderID and KeyInfo Elements	64
10.3	HTTPS+SAML	64
10	0.3.1 Service Type and Service Media Type	64
10	0.3.2 Protocol	65
11 Pr	oxy 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 Re	edirect 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	2.5.1 Redirect Examples	77
12	2.5.2 Ref Examples	81
12.6	Recursion and Backtracking	84
13 Se	ervice Endpoint Selection	85
13.1	Processing Rules	85
13.2	Service Endpoint Selection Logic	87
13.3	Selection Element Matching Rules	88
13	3.3.1 Selection Element Match Options	88
13	3.3.2 The Match Attribute	88
13	3.3.3 Absent Selection Element Matching Rule	89
13	3.3.4 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	4 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	5 Status Codes and Error Processing	106
	15.1 Status Elements	106
	15.2 Status Codes	
	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	6 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	7 Extensibility and Versioning	112
	17.1 Extensibility	
	17.1.1 Extensibility of XRDs	
	17.1.2 Other Points of Extensibility	

	17.2 Versioning	113
	17.2.1 Version Numbering	113
	17.2.2 Versioning of the XRI Resolution Specification	113
	17.2.3 Versioning of Protocols	114
	17.2.4 Versioning of XRDs	114
18	Security and Data Protection	115
	18.1 DNS Spoofing or Poisoning	115
	18.2 HTTP Security	115
	18.3 SAML Considerations	115
	18.4 Limitations of Trusted Resolution	115
	18.5 Synonym Verification	116
	18.6 Redirect and Ref Management	116
	18.7 Community Root Authorities	116
	18.8 Caching Authorities	116
	18.9 Recursing and Proxy Resolution	116
	18.10 Denial-Of-Service Attacks	116
A.	Acknowledgments	117
B.	RelaxNG Schema for XRDS and XRD	118
C.	XML Schema for XRDS and XRD	
D.	Media Type Definition for application/xrds+xml	125
E.	Media Type Definition for application/xrd+xml	126
F.	Example Local Resolver Interface Definition	127
G.	Revision History	131

# **Table of Figures**

Figure 1: Four typical scenarios for XRI authority resolution	13
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	ı64
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	
Table 27: Match options for service endpoints.	92
Table 28: Values of the append 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.

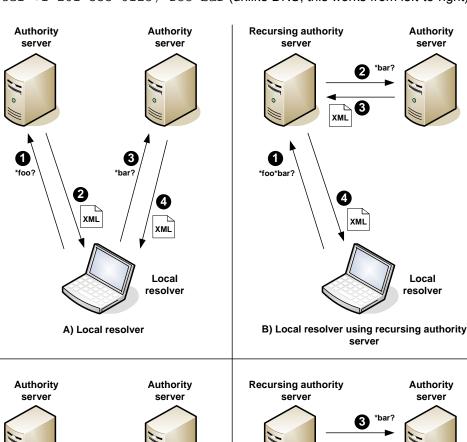
A simple non-recursing resolver will rely on a recursing nameserver to do this work. For example, it will send a query for the fully qualified domain name docs.oasis-open.org to a local nameserver. If the nameserver doesn't have the answer cached, it will resolve the domain name and return the results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

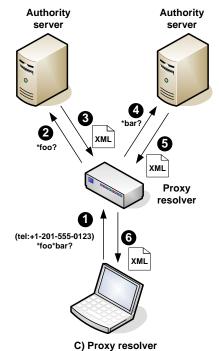
XRI resolution follows this same architecture except at a higher level of abstraction, i.e., rather than using UDP to resolve a domain name into a text-based resource descriptor, it uses HTTP(S) to resolve an XRI into an XML-based resource descriptor called an *XRDS document*. Table 1 provides a high-level comparison between DNS and XRI resolution architectures.

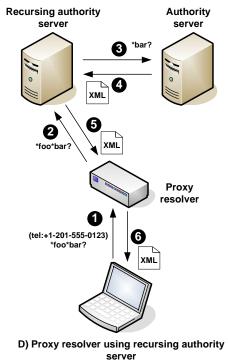
Resolution Component	DNS Architecture	XRI Architecture
Identifier	domain name	XRI (authority + path + query)
Resource record format	text (resource record)	XML (XRDS document)
Attribute identifier	string	anyURI
Network endpoint identifier	IP address	URI
Synonyms	CNAME	LocalID, EquivID, CanonicalID, CanonicalEquivID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

- 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
- implemented using a platform-specific API). Proxy resolvers enable applications—even those that 33
- only understand HTTP URIs—to easily access the functions of an XRI resolver remotely. 34
  - 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).







Authority

server

Local

resolver

31

- 39 Each of these scenarios may involve two phases of XRI resolution:
  - Phase 1: Authority resolution. This is the phase required to resolve the authority component of an XRI into an XRDS document describing the target authority. Authority resolution works iteratively from left-to-right across each subsegment in the authority component of the XRI. In XRIs, subsegments are delimited using either a specified set of symbol characters or parentheses. For example, in the XRI xri://(tel:+1-201-555-0123)\*foo\*bar, the authority subsegments are (tel:+1-201-555-0123) (the community root authority, in this case a URI expressed as an cross-reference delimited with parentheses), \*foo, (the first resolvable subsegment), and \*bar, (the second resolvable subsegment). Note that a resolver must be preconfigured (or have its own way of discovering) the community root authority starting point, so the community root subsegment is not resolved except in one special case (see section 9.1.6).
  - Phase 2: Optional service endpoint selection. Once authority resolution is complete, there is
    an optional second phase of XRI resolution to select a specific type of metadata from the final
    XRDS document retrieved called a service endpoint (SEP). Service endpoints are descriptors
    of concrete URIs at which network services are available for the target resource. Additional
    XRI resolution parameters as well as the path component of an XRI may be used as service
    endpoint selection criteria.
- 57 It is worth highlighting several other key differences between DNS and XRI resolution:
  - HTTP. As a resolution protocol, HTTP not only makes it easy to deploy XRI resolution services (including proxy resolution services), but also allows them to employ both HTTP security standards (e.g., HTTPS) and XML-based security standards (e.g., SAML). Although less efficient than UDP, HTTP(S) is suitable for the higher level of abstraction represented by XRIs and can take advantage of the full caching capabilities of modern web infrastructure.
  - XRDS documents. This simple, extensible XML resource description format makes it easy to
    describe the capabilities of any XRI-, IRI-, or URI-identified resource in a manner that can be
    consumed by any XML-aware application (or even by non-XRI aware browsers via a proxy
    resolver).
    - Service endpoint descriptors. DNS can use NAPTR records to do string transformations into
      URIs representing network endpoints. XRDS documents have service endpoint descriptors—
      elements that describe the set of URIs at which a particular type of service is available. Each
      service endpoint may present a different type of data or metadata representing or describing
      the identified resource. Thus XRI resolution can serve as a lightweight, interoperable
      discovery mechanism for resource attributes available via HTTP(S), LDAP, UDDI, SAML,
      WS-Trust, or other directory or discovery protocols.
  - Synonyms. DNS uses the CNAME attribute to establish equivalence between domain names.
     XRDS architecture supports four synonym elements (LocalID, EquivID, CanonicalID, and
     CanonicalEquivID) to provide robust support for mapping XRIs, IRIs, or URIs to other XRIs,
     IRIs, or URIs that identify the same target resource. This is particularly useful for discovering
     and mapping to persistent identifiers as often required by trust infrastructures.
  - Redirects and Refs. XRDS architecture also includes two mechanisms for distributed XRDS document management. The Redirect element allows an identifier authority to manage multiple XRDS documents describing a target resource from different network locations. The Ref element allows one identifier authority to delegate all or part of an XRDS document to a different identifier authority.

# 1.2 Structure of this Specification

85

107

- This specification is structured into the following sections:
- Conformance (section 2) specifies the conformance targets and conformance claims for this specification.
- Namespaces (section 3) specifies the XRI and XML namespaces and media types used for the XRI resolution protocol.
- 91 The next three sections cover XRDS documents and the requirements for XRDS clients and 92 servers:
- XRDS Documents (section 4) specifies a simple, flexible XML-based container for XRI resolution metadata, service endpoints, and/or other metadata describing a resource.
- XRDS Synonyms (section 5) specifies usage of the four XRDS synonym elements.
- Discovering an XRDS Document from an HTTP(S) URI (section 6) specifies a protocol for obtaining an XRDS description of a resource by starting from an HTTP(S) URI identifying the resource.
- The remaining sections cover XRI resolution and the requirements for XRI authority servers, local resolvers, and proxy resolvers:
- XRI Resolution Flow (section 7) provides a top-level flowchart of the XRI resolution function together with a list of other supporting flowcharts used throughout the specification.
- Inputs and Outputs (section 8) specifies the input parameters, output formats, and associated processing rules.
- Generic Authority Resolution (section 9) specifies a simple resolution protocol for the authority component of an XRI using HTTP/HTTPS as a transport.
  - Trusted Authority Resolution (section 10) specifies three extensions to generic authority resolution for creating a chain of trust between the participating identifier authorities using HTTPS connections, SAML assertions, or both.
- Proxy Resolution (section 11) specifies an HTTP(S) interface for an XRI resolver plus a format for expressing an XRI as an HTTP(S) URI to provide backwards compatibility with existing HTTP(S) infrastructure.
- Redirect and Ref Processing (section 12) specifies how a resolver follows a reference from one XRDS document to another to enable federation of XRDS documents across multiple network locations (Redirects) or identifier authorities (Refs).
- Service Endpoint Selection (section 13) specifies an optional second phase of resolution for
   selecting a set of service endpoints from an XRDS document.
- Synonym Verification (section 14) specifies how a resolver can verify that one XRI, IRI, or HTTP(S) URI is an authorized synonym for another.
- Status Codes and Error Processing (section 15) specifies status reporting and error handling.
- Use of HTTP(S) (section 16) specifies how the XRDS and XRI resolution protocols leverage features of the HTTP(S) protocol.
- Extensibility and Versioning (section 17) describes how the XRI resolution protocol can be easily extended and how new versions will be identified and accommodated.
- Security and Data Protection (section 18) summarizes key security and privacy
   considerations for XRI resolution infrastructure.

# 1.3 Terminology and Notation

- 128
- The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "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
- 131 in this document, they are meant in their natural language sense.
- 132 This specification uses the Augmented Backus-Naur Form (ABNF) syntax notation defined in
- 133 [RFC42341.

127

146

- 134 Other terms used in this document and not defined herein are defined in the glossary in Appendix
- 135 C of [XRISyntax].
- 136 Formatting conventions used in this document:
- 137 Examples look like this.
- 138 ABNF productions look like this.
- 139 In running text, XML elements, attributes, and values look like this.

#### 1.4 Examples 140

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

# 1.5 Normative References

147 148	[DNSSEC]	D. Eastlake, <i>Domain Name System Security Extensions</i> , http://www.ietf.org/rfc/rfc2535, IETF RFC 2535, March 1999.
149 150 151	[RFC2045]	N. Borenstein, N. Freed, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, http://www.ietf.org/rfc/rfc2045.txt, IETF RFC 2045, November 1996.
152 153 154	[RFC2046]	N. Borenstein, N. Freed, <i>Multipurpose Internet Mail Extensions (MIME)</i> Part Two: Media Types, http://www.ietf.org/rfc/rfc2046.txt, IETF RFC 2046, November 1996.
155 156	[RFC2119]	S. Bradner, Key Words for Use in RFCs to Indicate Requirement Levels, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
157 158	[RFC2141]	R. Moats, <i>URN Syntax</i> , http://www.ietf.org/rfc/rfc2141.txt, IETF RFC 2141, May 1997.
159 160 161	[RFC2483]	M. Meallling, R. Daniel Jr., <i>URI Resolution Services Necessary for URN Resolution</i> , http://www.ietf.org/rfc/rfc2483.txt, IETF RFC 2483, January 1999.
162 163 164	[RFC2616]	R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, <i>Hypertext Transfer Protocol HTTP/1.1</i> , http://www.ietf.org/rfc/rfc2616.txt, IETF RFC 2616, June 1999.
165 166	[RFC2818]	E. Rescorla, <i>HTTP over TLS</i> , http://www.ietf.org/rfc/rfc2818.txt, IETF RFC 2818, May 2000.
167 168	[RFC3023]	M. Murata, S. St.Laurent, D. Kohn, <i>XML Media Types</i> , http://www.ietf.org/rfc/rfc3023.txt, IETF RFC 3023, January 2001.
169 170 171	[RFC3986]	T. Berners-Lee, R. Fielding, L. Masinter, <i>Uniform Resource Identifier</i> ( <i>URI</i> ): Generic Syntax, http://www.ietf.org/rfc/rfc3986.txt, IETF RFC 3986, January 2005.

172 173	[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.
174 175 176	[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.
177 178 179 180	[SAML]	S. Cantor, J. Kemp, R. Philpott, E. Maler, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip, March 2005.
181 182 183 184 185	[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.
186 187	[UUID]	Open Systems Interconnection – <i>Remote Procedure Call</i> , ISO/IEC 11578:1996, http://www.iso.org/, August 2001.
188 189 190	[XML]	T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML) 1.0, Third Edition, World Wide Web Consortium, http://www.w3.org/TR/REC-xml/, February 2004.
191 192 193	[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/xmldsig-core/, February, 2002.
194 195 196	[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.
197 198 199	[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.
200 201 202	[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.
203 204 205	[XRIMetadata]	D. Reed, Extensible Resource Identifier (XRI) Metadata V2.0, http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf, March 2005.
206 207 208	[XRISyntax]	D. Reed, D. McAlpin, <i>Extensible Resource Identifier (XRI) Syntax V2.0</i> , http://docs.oasis-open.org/xri/xri-syntax/2.0/specs/cs01/xri-syntax-V2.0-cs.pdf, November 2005.
209 1	.6 Non-Norma	tive References
210 211	[XRIFAQ]	OASIS XRI Technical Committee, XRI 2.0 FAQ, http://www.oasis-open.org/committees/xri/faq.php, Work-In-Progress, March 2006.
212 213 214 215	[XRIReqs]	G. Wachob, D. Reed, M. Le Maitre, D. McAlpin, D. McPherson, Extensible Resource Identifier (XRI) Requirements and Glossary v1.0, http://www.oasis-open.org/committees/download.php/2523/xri-requirements-and-glossary-v1.0.doc, June 2003.
216 217	[WikipediaXRI]	Wikipedia entry on XRI (Extensible Resource Identifier), http://en.wikipedia.org/wiki/XRI, Wikipedia Foundation.
218	[Yadis]	J. Miller, <i>Yadis Specification Version 1.0</i> , http://yadis.org/, March 2006.

# 2 Conformance

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

# 2.1 Conformance Targets

- 223 The conformance targets of this specification are:
  - XRDS clients, which provide a limited subset of the functionality of XRI resolvers.
- 22. XRDS servers, which provide a limited subset of the functionality of XRI authority servers.
  - XRI local resolvers, which may implement any combination of the generic, HTTPS, or SAML resolution protocols.
  - XRI proxy resolvers, which may implement any combination of the generic, HTTPS, or SAML resolution protocols.
    - 5. XRI authority servers, which may implement any combination of the generic, HTTPS, or SAML resolution protocols.
- Note that a single implementation may serve any combination of these functions. For example, an
- 233 XRI authority server may also function as an XRDS client and server and an XRI local and proxy
- 234 resolver.

219

222

224

226

227

228

229

230

231

235

244

247

#### 2.2 Conformance Claims

- 236 A claim of conformance with this specification MUST meet the following requirements:
- 1. It MUST state which conformance targets it implements.
- 238 2. If the conformance target is an XRI local resolver, XRI proxy resolver, or XRI authority server, it MUST state which resolution protocols are supported, i.e., generic, HTTPS, and/or SAML.

#### 241 2.3 XRDS Clients

- An implementation conforms to this specification as an XRDS client if it meets the following conditions:
  - 1. It MAY implement parsing of XRDS Documents as specified in section 4.
- It MUST implement the client requirements of the XRDS request protocol specified in section 6.

#### 2.4 XRDS Servers

- An implementation conforms to this specification as an XRDS server if it meets the following conditions:
- 1. It MUST produce valid XRDS Documents as specified in section 4.
- 25. It MUST implement the server requirements of the XRDS request protocol specified in section 6.

#### 2.5 XRI Local Resolvers

#### 254 **2.5.1 Generic**

253

264

269

270

278

- An implementation conforms to this specification as a generic local resolver if it meets the following conditions:
- 1. It parses XRDS documents as specified in section 4.
- 258 2. It processes resolution inputs and outputs as specified in section 8.
- 3. It implements the resolver requirements of the generic resolution protocol specified in section 9.
- 4. It implements the Redirect and Ref processing rules specified in section 12.
- 5. It implements the Service Endpoint Selection processing rules specified in section 13.
- 263 6. It implements the Synonym Verification processing rules specified in section 14.
  - 7. It implements the Status Code and Error Processing rules specified in section 15.
- 265 8. It follows the HTTP(S) usage recommendations specified in section 16.

#### 266 **2.5.2 HTTPS**

- An implementation conforms to this specification as an HTTPS local resolver if it meets all the requirements of a generic local resolver plus the following conditions:
  - 1. It implements the resolver requirements of the HTTPS trusted resolution protocol specified in section 10.1.

#### 271 **2.5.3 SAML**

- An implementation conforms to this specification as a SAML local resolver if it meets all the requirements of a generic local resolver plus the following conditions:
- It implements the resolver requirements of the SAML trusted resolution protocol specified
   in section 10.2.
- It SHOULD also meet the requirements of an HTTPS local resolver. This is STRONGLY
   RECOMMENDED for confidentiality of SAML interactions.

# 2.6 XRI Proxy Resolvers

#### 279 **2.6.1 Generic**

- An implementation conforms to this specification as a generic proxy resolver if it meets all the requirements of a generic local resolver plus the following conditions:
- 282 1. It implements the requirements for a proxy resolver specified in section 11.

#### 283 **2.6.2 HTTPS**

- An implementation conforms to this specification as a HTTPS proxy resolver if it meets all the requirements of a HTTPS local resolver plus the following conditions:
- 1. It implements the requirements for a HTTPS proxy resolver specified in section 11.

#### 287 **2.6.3 SAML**

- An implementation conforms to this specification as a SAML proxy resolver if it meets all the requirements of a SAML local resolver plus the following conditions:
- 290 1. It implements the requirements for a proxy resolver specified in section 11.

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

# 2.7 XRI Authority Servers

#### 294 **2.7.1 Generic**

293

297

299

307

308

312

313

314

315

- An implementation conforms to this specification as a generic authority server if it meets the following conditions:
  - It produces XRDS documents as specified in section 4.
- 298 2. It assigns XRDS synonyms as specified in section 5.
  - 3. It processes resolution inputs and outputs as specified in section 8.
- 300 4. It implements the server requirements of the generic resolution protocol specified in section 9.
- It implements the Status Code and Error Processing rules specified in section 15.
- 303 6. It follows the HTTP(S) usage recommendations specified in section 16.

#### 304 **2.7.2 HTTPS**

- An implementation conforms to this specification as an HTTPS authority server if it meets all the requirements of a generic authority server plus the following conditions:
  - It implements the server requirements of the HTTPS trusted resolution protocol specified in section 10.1.

#### 309 **2.7.3 SAML**

- An implementation conforms to this specification as an SAML authority server if it meets all the requirements of a generic authority server plus the following conditions:
  - 1. It implements the server requirements of the SAML trusted resolution protocol specified in section 10.2.
  - It SHOULD also meet the requirements of an HTTPS authority server. This is STRONGLY RECOMMENDED for confidentiality of SAML interactions.

#### 316 2.8 Extensions

- 317 The protocols and XML documents defined in this specification MAY be extended. To maintain
- interoperability, extensions MUST use the extensibility architecture specified in section 17.
- 319 Extensions MUST NOT be implemented in a manner that would cause them to be non-
- interoperable with implementations that do not implement the extensions.

# 321 **2.9 Language**

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

# 3 Namespaces

# 3.1 XRI Namespaces for XRI Resolution

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

324

325

330

#### 3.1.1 XRIs Reserved for XRI Resolution

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

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

333 Table 2: XRIs reserved for XRI resolution.

# 3.1.2 XRIs Assigned to XRI Resolution Service Types

335 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

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

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

340

336

# 3.2 XML Namespaces for XRI Resolution

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

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

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

# 3.3 Media Types for XRI Resolution

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

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

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

352

353

354

355 356

341

345346

347 348

349

360 361

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 performed	14.3

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

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

# 4 XRDS Documents

- 363 XRI resolution provides resource description metadata using a simple, extensible XML format
- 364 called an XRDS (Extensible Resource Descriptor Sequence) document. An XRDS document
- 365 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
- 367 extended to publish any form of metadata about the resources they describe.

# 368 4.1 XRDS and XRD Namespaces and Schema Locations

- 369 An XRDS document is intended to serve exclusively as an XML container document for XML
- 370 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,
- 372 redirect and ref, that are used to identify the resource described by the XRDS document.
- 373 Both are of type anyURI. Use of these attributes is defined in section 12.5.
- 374 The elements in the XRD schema are intended for generic resource description, including the
- 375 metadata necessary for XRI resolution. Since the XRD schema has simple semantics that may
- 376 evolve over time, the version defined in this specification uses the XML namespace
- 377 xri://xrd\*(xv\*2.0). This namespace is versioned using XRI version metadata as defined
- 378 in [XRIMetadata].
- The attributes defined in both the XRDS and XRD schemas are not namespace qualified. In order
- 380 to prevent conflicts, attributes defined in extensions MUST be namespace qualified.
- 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
- 383 XRDS document) to be versioned over time. See section 17.2 for more about versioning of the
- 384 XRD schema.
- 385 The locations of the normative RelaxNG schema files for an XRDS document and an XRD
- 386 element as defined by this specification are:
- xrds.rnc: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrds.mc
- xrd.rnc: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrd.mc
- 389 •

362

- 390 The following URIs will always reference the latest versions of these files:
- xrds.rnc: http://docs.oasis-open.org/xri/2.0/specs/xrds.rnc
- xrd.rnc: http://docs.oasis-open.org/xri/2.0/specs/xrd.rnc
- 393 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.

#### 4.2 XRD Elements and Attributes

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

400

```
402
          <XRDS xmlns="xri://$xrds" ref="xri://(tel:+1-201-555-0123)*foo">
403
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
404
                   <Query>*foo</Query>
405
                   <Status code="100"/>
406
                   <ServerStatus code="100"/>
407
                   <Expires>2005-05-30T09:30:10Z</Expires>
408
                   <ProviderID>xri://(tel:+1-201-555-0123)</providerID>
409
                   <LocalID>*baz</LocalID>
410
                   <EquivID>https://example.com/example/resource/</EquivID>
411
                   <CanonicalID>xri://(tel:+1-201-555-0123)!1234</CanonicalID>
412
                   <CanonicalEquivID>
413
                   xri://=!4a76!c2f7!9033.78bd
414
                   </CanonicalEquivID>
415
                   <Service>
416
                       <ProviderID>
417
                        xri://(tel:+1-201-555-0123)!1234
418
                       </ProviderID>
419
                       <Type>xri://$res*auth*($v*2.0)</Type>
420
                       <MediaType>application/xrds+xml</MediaType>
421
                       <URI priority="10">http://resolve.example.com</URI>
422
                       <URI priority="15">http://resolve2.example.com</URI>
423
                       <URI>https://resolve.example.com</URI>
424
                   </Service>
425
                   <Service>
426
                       <ProviderID>
427
                       xri://(tel:+1-201-555-0123)!1234
428
                       </ProviderID>
429
                       <Type>xri://$res*auth*($v*2.0)</Type>
430
                       <MediaType>application/xrds+xml;https=true</MediaType>
431
                       <URI>https://resolve.example.com</URI>
432
                   </Service>
433
                   <Service>
434
                       <Type match="null" />
435
                       <Path select="true">/media/pictures</path>
436
                       <MediaType select="true">image/jpeg</MediaType>
437
                       <URI append="path" >http://pictures.example.com</URI>
438
                   </Service>
439
                   <Service>
440
                       <Type match="null" />
441
                       <Path select="true">/media/videos</Path>
442
                       <MediaType select="true">video/mpeg</MediaType>
443
                       <URI append="path" >http://videos.example.com</URI>
444
                  </Service>
445
                   <Service>
446
                       <ProviderID> xri://!!1000!1234.5678</providerID>
447
                       <Type match="null" />
448
                       <Path match="default" />
449
                       <URI>http://example.com/local</URI>
450
                   </Service>
451
                   <Service>
452
                       <Type>http://example.com/some/service/v3.1</Type>
453
                       <URI>http://example.com/some/service/endpoint</URI>
454
                       <LocalID>https://example.com/example/resource/</LocalID>
455
                   </Service>
456
              </XRD>
457
          </XRDS>
```

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

460

## 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

462

467

468 469

470

471

472

473

474

475 476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491 492

493

494

495

496

497

498 499

500

501 502

503

504

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 xrds: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 as 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.

#### 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:xrd:ServerStatus

0 or 1 per xrd:XRD element. Used by an XRI authority server to report the status of a resolution request to an XRI resolver. See section 15.1. Attributes:

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

#### xrd:XRD/xrd:Expires

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

#### xrd:XRD/xrd:Redirect

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

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

#### xrd:XRD/xrd:Ref

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

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

#### 4.2.2 Trust Elements

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

#### xrd:XRD/xrd:ProviderID

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

#### xrd:XRD/saml:Assertion

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

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

### 4.2.3 Synonym Elements

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

#### xrd:XRD/xrd:LocalID

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

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

#### xrd:XRD/xrd:EquivID

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

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

#### xrd:XRD/xrd:CanonicalID

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

#### xrd:XRD/xrd:CanonicalEquivID

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

# **4.2.4 Service Endpoint Descriptor Elements**

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

- 589 IMPORTANT: Establishing unambiguous priority is especially important for service endpoints 590 because they are used to control the direction of authority resolution, the order of Redirect and 591 Ref processing, and the prioritization of the final service endpoint URIs selected (if any). See 592 section 4.3.3 for rules and recommendations about usage of the priority attribute.
- Note that to prevent processing conflicts, the XRD schema permits only one of these element types in a service endpoint: xrd:URI, xrd:Redirect, or xrd:Ref.

#### xrd:XRD/xrd:Service

595 596

597

598

599

600

601

602

603 604

605

606

607

608 609

610 611

612

613

614

615

616 617

618

619

620 621

622

623

624

625

626

627

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

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

#### xrd:XRD/xrd:Service/xrd:LocalID

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

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

#### xrd:XRD/xrd:Service/xrd:URI

0 more per xrd:XRD/xrd:Service element. Type xs:anyURI. Choice between this or the xrd:XRD/xrd:Service/xrd:Redirect or xrd:XRD/xrd:Service/xrd:Ref elements. If present, it indicates a transport-level URI for accessing the capability described by the parent Service element. For the service types defined for XRI resolution in section 3.1.2, this URI MUST be an HTTP or HTTPS URI. Other services may use other transport protocols. Attributes:

- priority (type xs:nonNegativeInteger). OPTIONAL. See section 4.3.3.
- append (type xs:enumeration). OPTIONAL. Governs construction of the final service endpoint URI as defined in section 13.7.

#### xrd:XRD/xrd:Service/xrd:Redirect

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

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

#### xrd:XRD/ xrd:Service/xrd:Ref

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

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

#### 4.2.5 Service Endpoint Trust Elements

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

#### xrd:XRD/xrd:Service/xrd:ProviderID

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

#### xrd:XRD/xrd:Service/ds:KeyInfo

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

#### **4.2.6 Service Endpoint Selection Elements**

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

#### xrd:XRD/xrd:Service/xrd:Type

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

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

#### xrd:XRD/xrd:Service/xrd:Path

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

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

#### xrd:XRD/xrd:Service/xrd:MediaType

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

- of a media type defined in **[RFC2046]**. See section 3.3 for the media types used in XRI resolution. See section 13.3.8 for MediaType element matching rules. Attributes:
- match (type xs:enumeration). OPTIONAL. See section 13.3.2.
- select (type xs:boolean). OPTIONAL. See section 13.4.2.
- The XRD schema (Appendix B) allows other elements and attributes from other namespaces to
- be added throughout. As described in section 17.1.1, these points of extensibility can be used to
- deploy new XRI resolution schemes, new service description schemes, or other metadata about
- the described resource.

681

682

# 4.3 XRD Attribute Processing Rules

#### 4.3.1 ID Attribute

- For uses such as SAML trusted resolution (section 10.2) that require unique identification of
- multiple XRD elements within an XRDS document, the XRD element uses the implicit xml:id
- attribute as defined by the W3C XML ID specification [XMLID]. Note that this attribute is NOT
- 686 explicitly declared in either the RelaxNG schema in Appendix B or the XML Schema in Appendix
- 687 C since it is inherently included by the extensibility design of both schemas.
- 688 If present, the value of this attribute MUST be unique for all elements in the containing XML
- document. Because an XRI resolver may need to assemble multiple XRDs received from different
- authority servers into one XRDS document, there MUST be negligible probability that the value of
- the xrd:XRD/@xml:id attribute is not globally unique. For this reason the value of this attribute
- 692 SHOULD be a UUID as defined by **[UUID]** prefixed by a single underscore character ("\_") in
- order to make it a legal NCName as required by [XMLID]. However, the value of this attribute
- MAY be generated by any algorithm that fulfills the same requirements of global uniqueness and
- 695 NCName conformance.
- Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their
- 697 XML document order MUST match the order in which they were resolved (see section 9.1.2).
- 698 Also, if Redirect or Ref processing requires the same XRD to be included in an XRDS document
- 699 twice (via a nested XRDS document), that XRD MUST reference the previous instance using the
- 700 xrd:XRD/@idref attribute as defined in section 12.5.

#### 701 4.3.2 Version Attribute

- 702 Unlike the XRDS element, which is not intended to be versioned, the xrd: XRD element has the
- optional attribute xrd: XRD/@version. Use of this attribute is REQUIRED for XRI resolution.
- 704 The value of this attribute MUST be the exact numeric version value of the XRI Resolution
- specification to which the containing XRD element conforms. See sections 3.1.1 and 17.2.1.
- General rules about versioning of the XRI resolution protocol are defined in section 17.2. Specific
- 707 rules for processing the XRD version attribute are specified in section 17.2.4.

# 4.3.3 Priority Attribute

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

- Following are the normative processing rules that apply whenever there is more than one instance of the same type of element selected in an XRD (if there is only one instance selected. the priority attribute is ignored.)
  - 1. The consuming application SHOULD select the element instance with the lowest numeric value of the priority attribute. For example, an element with priority attribute value of "10" should be selected before an element with a priority attribute value of "11", and an element with priority attribute value of "11" should be selected before an element with a priority attribute value of "25". Zero is the highest priority attribute value. Null is the lowest priority attribute value—it is the equivalent of a value of infinity. It is RECOMMENDED to use a large finite value (100 or more) rather than a null value.
  - 2. If an element has no priority attribute, its priority attribute value is considered to be null, i.e., the lowest possible priority value. Rather than omitting a priority attribute, it is RECOMMENDED that XRI authorities follow the standard practice in DNS and set the default priority attribute value to "10".
  - 3. If two or more instances of the same element type have identical priority attribute values (including the null value), the consuming application SHOULD select one of the instances at random. This consuming application SHOULD NOT simply choose the first instance that appears in XML document order.
  - IMPORTANT: It is vital that implementers observe the preceding rule in order to support intentional redundancy or load balancing semantics. At the same time, it is vital that XRDS authors understand that this rule can result in non-deterministic behavior if two or more of the same type of synonym elements or service endpoint elements are included with the same priority in an XRD but are NOT intended for redundancy or load balancing.
    - 4. An element selected according to these rules is referred to in this specification as the highest priority element. If this element is subsequently disqualified from the set of qualified elements, the next element selected according to these rules is referred to as the next highest priority element. If a resolution operation specifying selection of the highest priority element fails, the resolver SHOULD attempt to select the next highest priority element unless otherwise specified. This process SHOULD be continued for all other instances of the qualified elements until success is achieved or all instances are exhausted.

# 4.4 XRI and IRI Encoding Requirements

- 748 The W3C XML 1.0 specification [XML] requires values of XML elements of type xs:anyURI to be valid IRIs. Thus all XRIs used as the values of XRD elements of this type MUST be in at least 749
- 750 IRI-normal form as defined in section 2.3 of [XRISyntax].
- A further restriction applies to XRIs or IRIs used in XRI resolution because it relies on HTTP(S) as 751 752
- a transport protocol. Therefore when an XRI or IRI is used as the value of an xrd: Ouery,
- 753 xrd:LocalID, xrd:EquivID, xrd:CanonicalID, xrd:CanonicalEquivID,
- 754 xrd:Redirect, xrd:Ref, xrd:Type, or xrd:Path element, it MUST be in URI-normal form
- 755 as defined in section 2.3 of [XRISyntax].
- 756 Note: XRIs composed entirely of valid URI characters and which do not use XRI parenthetical
- 757 cross-reference syntax do not require escaping in the transformation to URI-normal form.
- 758 However, XRIs that use characters valid only in IRIs or that use XRI parenthetical cross-reference
- 759 syntax may require percent encoding in the transformation to URI-normal form as explained in
- 760 section 2.3 of [XRISyntax].

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

# 5 XRD Synonym Elements

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

765 Table 7: The four XRD synonym elements.

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

# 5.1 Query Identifiers

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

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

# 774 5.1.1 HTTP(S) URI Query Identifiers

- 775 If the fully-qualified query identifier is an absolute HTTP(S) URI, the XRDS document to which it
- 776 resolves (via the protocol specified in section 6) MUST contain a single XRD. This XRD MAY
- 777 include an xrd: Query element; if present, the value MUST be equivalent to the original HTTP(S)
- 778 URI query identifier.
- 779 In this single XRD, all synonym elements in Table 7 assert synonyms for the original HTTP(S)
- 780 URI.

761

762 763

764

767

771

772

773

# 781 5.1.2 XRI Query Identifiers

- 782 If the fully-qualifed query identifier is an absolute XRI, the XRDS document to which it resolves
- 783 (via the protocol specified in section 9.1.2) MAY contain multiple XRDs, each XRD corresponding
- 784 to one subsegment of the authority component of the XRI. Each XRD SHOULD include an
- 785 xrd: Query element that echos back the XRI subsegment described by this XRD. This is called
- the *local query identifier*, because it represents just one subsegment of the fully-qualifed query
- 787 identifier.
- 788 At any point in the XRI resolution chain, the combination of the community root authority XRI
- 789 (section 9.1.3) plus all local query identifiers resolved in all XRDs up to that point is called the
- 790 current fully-qualified query identifier. When the resolution chain is complete, the current fully-
- qualified query identifier is equal to the starting fully-qualifed query identifier.

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

# 5.2 Synonym Elements

#### 796 **5.2.1 LocalID**

795

- 797 In an XRD, a synonym for the local query identifier is asserted using the xrd:LocalID element.
- 798 LocalIDs may be used at both the XRD level (as a child of the root xrd: XRD element) and at the
- 799 service endpoint (SEP) level (as a child of the root xrd:XRD/xrd:Service element).
- 800 At the XRD level, the value of the xrd:XRD/xrd:LocalID element asserts a synonym that is
- 801 interchangeable with the contents of the xrd: Query element in the XRD. This means that
- resolution of a LocalID in the context of the same parent authority using the same resolution
- 803 guery parameters as the current query MUST result in an equivalent XRD as defined in section
- 5.4. It also means an XRI resolver MAY use a LocalID as an alternate key for the XRD in its
- 805 cache (see section 16.4.2).
- 806 If the parent authority has assigned a persistent local identifier to the resource described by an
- XRD, it SHOULD return this persistent identifier as an xrd:XRD/xrd:LocalID value in any
- 808 resolution response for a reassignable local identifier for the same resource. The reverse MAY
- also be true, however parent authorities MAY adopt privacy or other policies that restrict the
- reassignable synonyms returned for any particular resolution request.
- At the SEP level, the xrd:XRD/xrd:Service/xrd:LocalID element MAY be used to express
- 812 either a local or global identifier for the target resource in the context of the specific service being
- 813 described. If present, consuming applications SHOULD use the value of the highest priority
- 814 instance of the xrd:XRD/xrd:Service/xrd:LocalID element to identify the target resource
- 815 in the context of this service endpoint. If not present, consuming applications SHOULD select a
- 816 synonym as defined in section 5.6.
- 817 SPECIAL SECURITY CONSIDERATIONS: A parent authority SHOULD NOT permit a child
- authority to edit a LocalID value in an XRD without authenticating the child authority and verifying
- that the child authority is authorized to use this LocalID value either at the XRD level and/or the
- 820 SEP level.

828

829

830

831

## 821 **5.2.2 EquivID**

- 822 In an XRD, any synonym for the current fully-qualified query identifier except a CanonicalID or a
- 823 CanonicalEquivID (see below) is asserted using the xrd : EquivID element, Unlike a LocalID, an
- 824 EquivID is NOT REQUIRED to be issued by the parent authority.
- An EquivID MUST be an absolute identifier. For durability of the reference, it is RECOMMENDED
- to use a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].
- 827 An EquivID element is OPTIONAL in an XRD except in two cases:
  - 1. When it is REQUIRED as a backpointer to verify another EquivID element in a different XRD as specified in section 14.2.
  - 2. When it is REQUIRED as a backpointer to verify a CanonicalEquivID element as specified in section 14.3.3.
- 832 SPECIAL SECURITY CONSIDERATIONS: An EquivID synonym SHOULD NOT be trusted
- 833 unless it is verified. This function is not performed automatically by XRI resolvers but may be
- easily performed by consuming applications using one additional XRI resolution call as specified
- in section 14.2. A parent authority SHOULD NOT permit a child authority to edit the EquivID value
- in an XRD without authenticating the child authority and verifying that the child authority is

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

#### 5.2.3 CanonicalID

840

845

846

847

848 849

850

851

852 853

854

855

856 857

858

871 872

873

874

875

876

877 878

879 880

881

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

- 1. It MUST be an identifier for which the parent authority is the final authority. This means that resolution of a CanonicalID using the same resolution query parameters as the current query MUST result in an equivalent XRD as defined in section 5.4.
- 2. If the CanonicalID is any XRI except a community root authority XRI (section 9.1.3), it MUST consist of the parent authority's CanonicalID plus one additional subsegment. (In XRI resolution the parent authority's CanonicalID is always in the immediately preceding XRD in the same XRDS document, not in a nested XRDS document produced as a result of Redirect and Ref processing as defined in section 12.5.) For example, if the CanonicalID asserted for a target resource is @!1!2!3, then the CanonicalID for the parent authority must be @!1!2. See section 14.3.2 for details.
- 3. Once assigned, a parent authority SHOULD NEVER: a) change or reassign a CanonicalID value, or b) stop asserting a CanonicalID element in an XRD in which it has been asserted. For this reason, it is STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].
- As a best practice, a parent authority SHOULD ALWAYS publish a CanonicalID element in an XRD, even if its value is equivalent to the current fully-qualified query identifier. This practice:
- Makes it unambiguous to consuming applications which absolute synonym they should use to identify the target resource in the context of the parent authority.
- Enables child authorities to issue their own verifiable CanonicalIDs.
- Enables verification of a CanonicalEquivID if asserted (below).
- SPECIAL SECURITY CONSIDERATIONS: A CanonicalID synonym SHOULD NOT be trusted unless it is verified. CanonicalID verification is performed automatically during resolution by an XRI resolver unless this function is explicitly turned off; see section 14. A parent authority SHOULD NOT permit a child authority to edit the CanonicalID value in an XRD without authenticating the child authority and verifying that the child authority is authorized to use this CanonicalID value.

# 5.2.4 CanonicalEquivID

- The purpose of the xrd:CanonicalEquivID element is to assert a canonical synonym for the fully-qualified query identifier for which the parent authority MAY NOT be authoritative. A CanonicalEquivID MUST meet all the requirements of an EquivID plus the following:
  - 1. In order for the value of the xrd:CanonicalEquivID element to be verified: a) the XRD in which it appears MUST include a CanonicalID that can be verified as specified in section 14.2, and b) the XRD to which it resolves MUST meet the rules specified in section 14.3.3. In particular, those rules require that the CanonicalID of that XRD match the asserted CanonicalEquivID.
  - 2. For the same reasons as with a CanonicalID, it is STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

- 3. Although the CanonicalEquivID associated with a CanonicalID MAY change over time, at any one point in time, every XRD from the same parent authority that asserts the same CanonicalID value MUST assert the same CanonicalEquivID value if the XRD includes a CanonicalEquivID element.
- As a best practice, a parent authority SHOULD publish a CanonicalEquivID in an XRD if consuming applications SHOULD be able to persistently identify the target resource using this identifier in other contexts. Also, a CanonicalEquivID value SHOULD change very infrequently, if at all.
- SPECIAL SECURITY CONSIDERATIONS: A CanonicalEquivID synonym SHOULD NOT be trusted unless it is verified. Verification of the value of the CanonicalEquivID element in the final XRD in an XRDS document is performed automatically during resolution by an XRI resolver unless this function is explicitly turned off; see section 14. A parent authority SHOULD NOT permit a child authority to edit the CanonicalEquivID value in an XRD without authenticating the child authority and verifying that the child authority is authorized to use this CanonicalEquivID value.

#### 5.3 Redirect and Ref Elements

897

915

916

917 918

919

920

921

922

923

- While similar in some ways to synonym elements, the xrd:Redirect and xrd:Ref elements

  MUST NOT be used to assert a synonym. Instead their purpose is to assert that a different XRDS

  document is authorized to serve as an equally valid descriptor of the target resource. These
  elements enable separation of synonym assertion semantics vs. distributed XRDS document
  authorization semantics.
- In the same way as a LocalID, both a Redirect and a Ref may be used in an XRD at either the XRD level (as a child of the root xrd:XRD element) and at the SEP level (as a child of the root xrd:XRD/xrd:Service element). The complete rules for Redirect and Ref processing in XRI resolution are specified in section 12.
- 907 If two independent resources are later merged into the same resource, e.g., two businesses are 908 merged into one, the use of an EquivID, CanonicalID, or CanonicalEquivID element SHOULD be 909 combined with the use of a Redirect or Ref element to provide the semantics of BOTH identifier 910 synonymity and XRDS authorization.
- 911 SPECIAL SECURITY CONSIDERATIONS: A parent authority SHOULD NOT permit a child 912 authority to edit a Redirect or Ref value in an XRD without authenticating the child authority and 913 verifying that the child authority is authorized to use this Redirect or Ref value at either the XRD 914 level and/or the SEP level.

# 5.4 XRD Equivalence

- LocalID and CanonicalID synonyms are required to resolve to an XRD that is equivalent to the XRD in which the synonym is asserted. Two XRDs MUST be considered equivalent if they meet the following rules:
  - 1. Both XRDs contain a CanonicalID element.
  - 2. The values of these CanonicalID elements are equivalent according to the equivalence rules of the applicable identifier scheme. Note that these identifiers MUST be in URI-normal form as specified in section 4.4. In addition, if the CanonicalID values are HTTP(S) URIs, fragments MUST be considered significant in comparison.

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

# 5.5 Synonym Verification

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

928

932

937

938

939

# 5.6 Synonym Selection

- It is out of the scope of this specification to specify policies consuming applications should use to select their desired synonym(s) to identify a target resource. However, the following are
- 935 RECOMMENDED best practices:
- 936 Only select a verified synonym (see above).
  - Select a persistent synonym, particularly if a long term or immutable reference is required. If a persistent synonym is present, other reassignable synonyms (including the current fullyqualified query identifier) SHOULD be treated only as temporary identifiers.
- Select a CanonicalID if present, verified, and persistent. This identifier SHOULD be used
   whenever referencing the target resource in the context of the parent authority issuing the
   CanonicalID.
- If possible, *also* select a CanonicalEquivID if present, verified, and persistent. This identifier SHOULD be used as a reference to the target resource in any context other than that of the parent authority.
- When selecting a synonym to use in the context of a specific service endpoint, follow the recommendations for use of the xrd:XRD/xrd:Service/xrd:LocalID element as specified in section 5.2.1.

# 949 6 Discovering an XRDS Document from an HTTP(S) URI

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

954 XRDS document starting with an HTTP(S) URI.

#### 6.1 Overview

955 956

957 958

959

960

982

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

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

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

#### 964 **6.2 HEAD Protocol**

- Under this protocol the XRDS client MUST begin by issuing an HTTP(S) HEAD request. This request SHOULD include an Accept header specifying the content type application/xrds+xml.
- The response from the XRDS server MUST be HTTP(S) response-headers only, which MAY include one or both of the following:
- 970 1. An X-XRDS-Location response-header.
- 971 2. A content type response-header specifying the content type application/xrds+xml.
- 972 If the response includes the first option above, the value of the X-XRDS-Location response-973 header MUST be an HTTP(S) URI which gives the location of an XRDS document describing the 974 target resource. The XRDS client MUST then request this document as specified in section 6.3.
- 975 If the response includes the second option above, the XRDS client MUST request the XRDS
- 976 document from the original HTTP(S) URI as specified in section 6.3.
- 977 If the response includes both options above, the value of the  ${\tt X-XRDS-Location}$  element in the
- 978 HTTP(S) response-header MUST take precedence.
- 979 If response includes neither of the two options above, this protocol fails and the XRDS client
- 980 MUST fall back to using the protocol specified in section 6.3.
- 981 In all cases the HTTP(S) status messages and error codes defined in [RFC2616] apply.

#### 6.3 GET Protocol

- Under this protocol the XRDS client MUST begin by issuing an HTTP(S) GET request. This
- 984 request SHOULD include an Accept header specifying the content type
- 985 application/xrds+xml.
- 986 The XRDS server response MUST be one of four options:
- 987 1. HTTP(S) response-headers only as defined in section 6.2.

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

990

991

992

993 If the response is only HTTP(S) response headers as defined in section 6.2, or if in addition to 994 these response headers it includes any document other than the two document types defined in 995 the third and fourth options above, the protocol MUST proceed as defined in section 6.2, except 996 that there is no fallback to this section if that protocol fails.

- 1004 If the response includes both an HTTP(S) response header and the HTML document defined in the third option above, the value of the X-XRDS-Location element in the HTTP(S) response-header MUST take precedence.
- 1007 If the response includes an XRDS document as specified in the fourth option above, the protocol has completed successfully.
- 1009 In all cases the HTTP(S) status messages and error codes defined in [RFC2616] apply.
- Note: If the XRDS server supports content negotiation, the response SHOULD include a Vary:
- header to allow caches to properly interpret future requests. This header SHOULD be present
- even in the case where the HTML page is returned (instead of an XRDS document).

## 7 XRI Resolution Flow

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*.

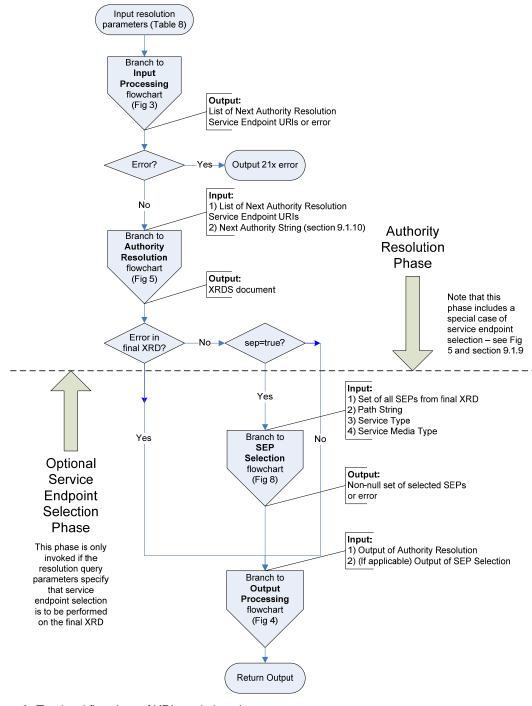


Figure 2: Top-level flowchart of XRI resolution phases.

10131014

1015

1016

- Branches of this top-level flowchart are used throughout the specification to provide a logical overview of key components of XRI resolution. The branch flowcharts include:
- Figure 3: Input processing (section 8.1).
- Figure 4: Output processing (section 8.2).
- Figure 5: Authority resolution (section 9).
- Figure 6: XRDS requests (section 9.1.3).
- Figure 7: Redirect and Ref processing (section 12).
- Figure 8: Service endpoint selection (section 13).
- Figure 9: Service endpoint selection logic (section 13.2).
- 1029 IMPORTANT: In all cases the flowcharts are informative and the specification text is normative.
  1030 However, the flowcharts are recommended as an aid in reading the specification. In particular,
  1031 those highlighted in **bold** above illustrate the recursive calls for authority resolution and service
- 1032 endpoint selection used during Redirect and Ref processing (section 12). Implementers should
- pay special attention to these calls and the guidance in section 12.6, *Recursion and Backtracking*.

# 8 Inputs and Outputs

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.

## 8.1 Inputs

1034

1035 1036

1037

1038

10391040

1041 1042

1043

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	Туре	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

1044 Table 8: Input parameters for XRI resolution.

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

- 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.

1058

1045

1046

1047

1048

1049 1050

1051

1052

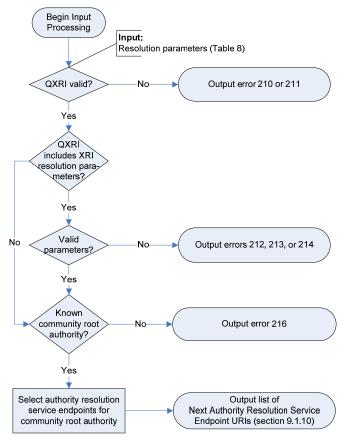
1053

1054

1055

1056

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



1060 1061

Figure 3: Input processing flowchart.

1062 1063 The following sections specify additional validation and usage requirements that apply to particular input parameters.

## 1065 8.1.1 QXRI (Authority String, Path String, and Query String)

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

Logical Parameter Name	Туре	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.

1068 Table 9: Subparameters of the QXRI input parameter.

1066

1067

1073

1074

1075

1076

1077 1078

1079

1080

1081 1082

1083

1086

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

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

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

#### 8.1.2 Resolution Output Format

- The Resolution Output Format is an OPTIONAL parameter that, together with its subparameters, is used to specify:
  - The media type for the resolution response.
- Whether generic or trusted resolution must be used by the resolver.
- Whether Refs should be followed during resolution.
- Whether CanonicalID verification should not be performed during resolution.
- Whether service endpoint selection should be performed on the final XRD.

- Whether default matches should be ignored during service endpoint selection.
- Whether URIs should automatically be constructed in the final XRD.
- 1093 Following are the normative requirements for the use of this parameter.
  - 1. The Resolution Output Format MUST be one of the values specified in Table 5 and MAY include any of the subparameters specified in Table 6.
  - 2. If the value of the https subparameter is TRUE, the resolver MUST use the HTTPS trusted authority resolution protocol specified in section 10.1 (or return an error indicating this is not supported).
  - 3. If the value of the saml subparameter is TRUE, the resolver MUST use the SAML trusted authority resolution protocol specified in section 10.2 (or return an error indicating this is not supported).
  - 4. If the value of both the https and saml subparameters are TRUE, the resolver MUST use the HTTPS+SAML trusted authority resolution protocol specified in section 10.3 (or return an error indicating this is not supported).
  - 5. If the value of the cid subparameter is TRUE or null, or if the parameter is absent, the resolver MUST perform CanonicalID verification as specified in section 14.3. If the value of the cid subparameter is FALSE, the resolver MUST NOT perform CanonicalID verification.
  - 6. If the value of the refs subparameter is TRUE or null, or if the parameter is absent, the resolver MUST perform Ref processing as specified in section 12. If the value of the refs subparameter is FALSE, the resolver MUST NOT perform Ref processing and must return an error if a Ref is encountered as specified in section 12.
  - 7. If the value of the sep subparameter is TRUE, the resolver MUST perform service endpoint selection on the final XRD (even if the values of all service endpoint selection parameters are null). If the value of the sep subparameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT perform service endpoint selection on the final XRD unless it is required to produce a URI List or HTTP(S) redirect. See section 8.2.
  - 8. If the value of the nodefault\_t, nodefault\_p, or nodefault\_m subparameter is TRUE, the resolver MUST ignore default matches on the corresponding service endpoint selection element categories as specified in section 13.3.2.
  - 9. If the value of the uric subparameter is TRUE, the resolver MUST perform service endpoint URI construction as specified in section 13.7.1. If the value of the uric subparameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT perform service endpoint URI construction.
- Future versions of this specification, or other specifications for XRI resolution, MAY use other values for Resolution Output Format or its subparameters.

#### 8.1.3 Service Type

- 1128 The Service Type is an OPTIONAL value of type xs:anyURI used to request a specific type of
- 1129 service in the service endpoint selection phase (section 11). The value of this parameter MUST
- 1130 be a valid absolute XRI, IRI, or URI in URI-normal form as defined by [XRISyntax]. (Note that
- 1131 URI-normal form is required so this parameter may be passed to a proxy resolver in a QXRI
- 1132 query parameter as defined in section 11.) The Service Type values defined for XRI resolution
- 1133 services are specified in section 3.1.2. The rules for matching the value of the Service Type
- 1134 parameter to the value of the xrd: XRD/xrd: Service/xrd: Type element are specified in
- 1135 section 13.3.6.

1094

1095 1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106 1107

1108

1109

1110

1111

1112

1113

1114

1115

1116 1117

1118

1119

1120

1121

1122 1123

1124

## 8.1.4 Service Media Type

- 1137 The Service Media Type is an OPTIONAL string used to request a specific media type in the 1138 service endpoint selection phase (section 11). The value of this parameter MUST be a valid media type as defined by [RFC2046]. The Service Media Type values defined for XRI resolution 1139
- services are specified in section 3.3. The rules for matching the value of the Service Media Type 1140
- 1141 parameter to the value of the xrd:XRD/xrd:Service/xrd:MediaType element are specified
- 1142 in section 13.3.8.

1136

1143

## 8.2 Outputs

- 1144 Table 10 summarizes the logical outputs of XRI resolution. Note that these are defined in terms of
- 1145 media types returned by authority servers and proxy resolvers. A local resolver API MAY
- 1146 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	null

1147 Table 10: Outputs of XRI resolution.

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

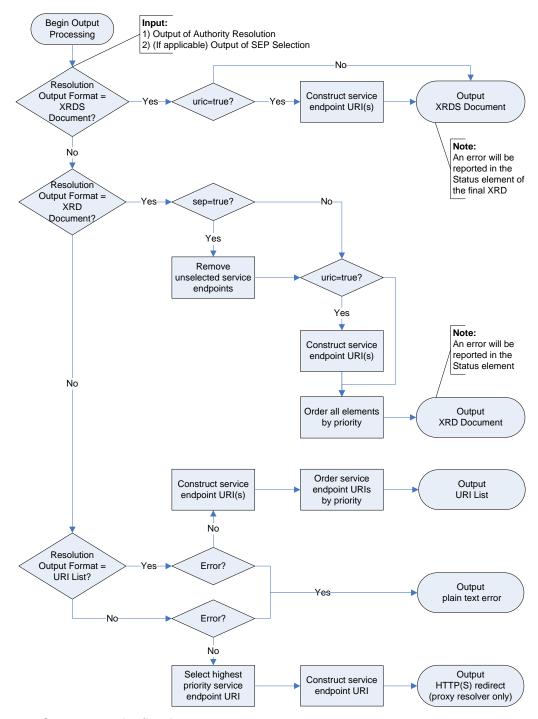


Figure 4: Output processing flowchart.

11521153

1155

1154 The following sections provide additional construction and validation requirements.

#### 8.2.1 XRDS Document

- 1157 If the value of the Resolution Output Format parameter is application/xrds+xml, the following rules apply.
  - 1. The output MUST be a valid XRDS document according to the schema defined in Appendix B.
  - 2. The XRDS document MUST contain an ordered list of xrd:XRD elements—one for each authority subsegment successfully resolved by the resolver client. This list MUST appear in the same order as the corresponding subsegments in the Authority String.
  - 3. Each of the contained XRD elements must be a valid XRD element according to the schema defined in Appendix B.
  - 4. The XRD elements MUST conform to the additional requirements in section 4.
  - 5. If the value of the saml subparameter of the Resolution Output Format is TRUE, the XRD elements MUST conform to the additional requirements in section 10.2.
    - 6. If Redirect or Ref processing is necessary during the authority resolution or service endpoint selection process, it MUST result in a valid nested XRDS document as defined in section 12.
    - 7. If the value of the sep subparameter is TRUE, service endpoint selection MUST be performed as defined in section 13, even if the values of all three service endpoint selection input parameters (Service Type, Path String, and Service Media Type) are null.
  - IMPORTANT: No filtering of the final XRD is performed when returning an XRDS document. Filtering is only performed when the requested Resolution Output Format is an XRD element see the next section.
    - 8. If the value of the cid subparameter is TRUE, synonym verification MUST be reported using the xrd:Status element of each XRD in the XRDS document as defined in section 14.
    - 9. If the output is an error, this error MUST be returned using the xrd:Status element of the final XRD in the XRDS document as defined in section 15.

#### 8.2.2 XRD Element

- If the value of the Resolution Output Format parameter is application/xrd+xml, the following rules apply.
  - The output MUST be a valid XRD element according to the schema defined in Appendix B.
  - 2. The XRD elements MUST conform to the additional requirements in section 4.
  - 3. If the value of the saml subparameter of the Resolution Output Format is TRUE, the XRD element MUST conform to the additional requirements in section 10.2.
  - 4. If the value of the sep subparameter is FALSE or null, or if this parameter is absent, the XRD MUST be the final XRD in the XRDS document produced as a result of authority resolution. Service endpoint selection or any other filtering of the XRD element MUST NOT be performed.
  - 5. If the value of the sep subparameter is TRUE, service endpoint selection MUST be performed as defined in section 13, even if the values of all service endpoint selection input parameters are null.
  - 6. If service endpoint selection is performed, the only xrd:Service elements in the XRD element MUST be those selected according to the rules specified in section 13. If no service endpoints were selected by those rules, no xrd:Service elements will be

- present. In addition, all elements within the XRD element that are subject to the global priority attribute (even if the attribute is absent or null) MUST be returned in order of highest to lowest priority as defined in section 4.3.3.
- 1204 IMPORTANT: Any other filtering of the XRD element MUST NOT be performed. Note that this
  1205 means that if the XRD element includes a SAML signature element as defined in section 10.2,
  1206 this element is still returned inside the XRD element even though it may not be able to be verified
  1207 by a consuming application.
- 7. If the value of the cid subparameter is TRUE, synonym verification MUST be reported using the xrd:Status element of each XRD in the XRDS document as defined in section 14.
- 1211 8. If the output is an error, this error MUST be returned using the xrd:Status element as defined in section 15.

## 8.2.3 **URI List**

1213

1216

1217

1218

1219 1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1234

1235

- 1214 If the value of the Resolution Output Format parameter is text/uri-list, the following rules apply.
  - 1. For this output, service endpoint selection is REQUIRED, even if the values of all service endpoint selection input parameters are null.
  - 2. If authority resolution and service endpoint selection are both successful, the output MUST be a valid URI List as defined by section 5 of [RFC2483].
  - 3. If, after applying the service endpoint selection rules, more than one service endpoint is selected, the highest priority xrd:XRD/xrd:Service element MUST be selected as defined in section 4.3.3.
  - 4. If the final selected xrd:XRD/xrd:Service element contains a xrd:XRD/xrd:Service/xrd:Redirect or xrd:XRD/xrd:Service/xrd:Ref element, Redirect and Ref processing MUST be performed as described in section 12. This rule applies iteratively to each new XRDS document resolved.
  - 5. From the final selected xrd:XRD/xrd:Service element, the service endpoint URI(s) MUST be constructed as defined in section 13.7.1.
  - 6. The URIs MUST be returned in order of highest to lowest priority of the source xrd:URI elements within the selected xrd:Service element as defined in section 4.3.3. When two or more of the source xrd:URI elements have equal priority, their constructed URIs SHOULD be returned in random order.
- 1233 IMPORTANT: Any other filtering of the URI list MUST NOT be performed.
  - 7. If the output is an error, it MUST be returned with the content type text/plain as defined in section 15.

## 1236 **8.2.4 HTTP(S) Redirect**

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

# 9 Generic Authority Resolution Service

- 1240 As discussed in section 1.1 and illustrated in Figure 2, authority resolution is the first phase of XRI
- 1241 resolution. This phase applies only to resolving the subsegments in the Authority String of the
- 1242 QXRI. The Authority String may identify either an XRI authority or an IRI authority as described in
- 1243 section 2.2.1 of [XRISyntax].

1239

1249

1250

1266

- 1244 XRI authorities and IRI authorities have different syntactic structures, partially due to the higher
- 1245 level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved
- 1246 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.

## 9.1 XRI Authority Resolution

## 9.1.1 Service Type and Service Media Type

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)

- 1252 Table 11: Service Type and Service Media Type values for generic authority resolution.
- 1253 A generic authority resolution service endpoint advertised in an XRDS document MUST use the 1254 Service Type identifier and MAY use the Service Media Type identifier defined in Table 11.
- 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:

1260 application/xrds+xml

1261 application/xrds+xml;trust=none

1262 application/xrds+xml;https=false

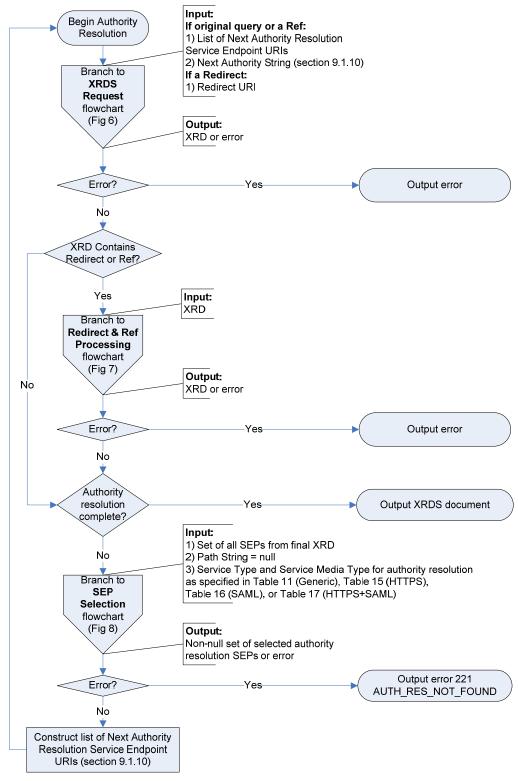
1263 application/xrds+xml;saml=false

1264 application/xrds+xml;https=false;saml=false

1265 application/xrds+xml;saml=false;https=false

### 1267 **9.1.2 Protocol**

1268 Figure 5 (non-normative) illustrates the overall logical flow of generic authority resolution.



1270 Figure 5: Authority resolution flowchart.

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

1273

1274

1275

1276

1277

1278

1279 1280

1281

1282

1283

1284

1285

1286 1287

1288

1289

1290 1291

1292

1293

1294

1295

1296

1297

- 1. Each request for an XRDS document using HTTP(S) MUST conform to the requirements in section 9.1.3.
  - 2. For errors in XRDS document resolution requests, a resolver MUST implement failover handling as specified in section 9.1.4.
  - 3. The resolver MUST be preconfigured with or have a means of obtaining the XRDS document describing the community root authority for the XRI to be resolved as defined in section 9.1.5.
  - 4. The resolver MAY obtain the XRDS document describing the community root authority by requesting a self-describing XRDS document as defined in section 9.1.6.
  - 5. Resolution of each subsegment in the Authority String after the community root subsegment MUST proceed in subsegment order (left-to-right) using fully qualified subsegment values as defined in section 9.1.7.
  - Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as defined in section 9.1.8.
  - 7. For each iteration of the authority resolution process, the next authority resolution service endpoint MUST be selected as specified in section 9.1.9.
  - 8. For each iteration of the authority resolution process, an HTTP(S) URI (called the Next Authority URI) MUST be constructed according to the algorithm specified in section 9.1.10.
  - A resolver MAY request that a recursing authority server perform resolution of multiple subsequents as defined in section 9.1.11.
  - 10. For each iteration of the authority resolution process, a resolver MUST perform Redirect and Ref processing as specified in section 12. Note that if Redirect and Ref processing is successful, it will result in a nested XRDS document as specified in section 12.5 and illustrated in Figure 6.

## 9.1.3 Requesting an XRDS Document using HTTP(S)

1299

1300 Figure 6 (non-normative) illustrates the logical flow for requesting an XRDS document.

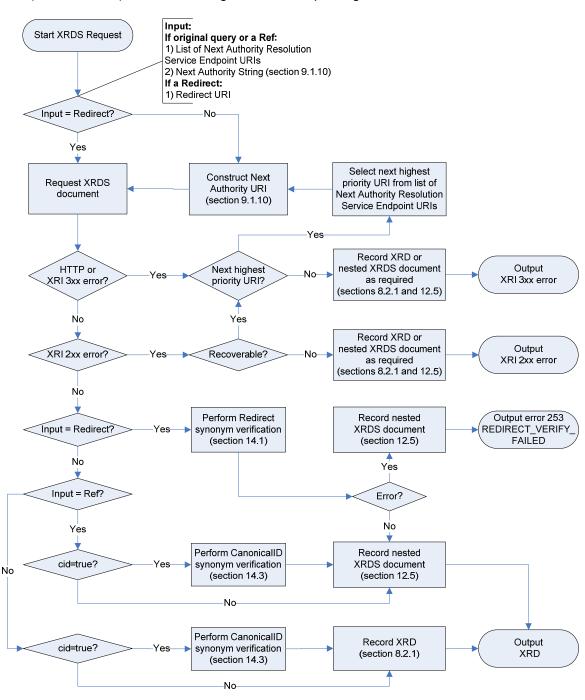


Figure 6: XRDS request flowchart.

1301 1302 1303

1304

1305 1306 Note that the term "Record" in Figure 6 means that if the Resolution Output Format is an XRDS document, this is the logical operation of appending either an XRD or an XRDS document at the proper nesting level within that output. See the examples in section 12.5.

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

## 9.1.4 Failover Handling

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319 1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1343

1344

1353

1354

- XRI infrastructure has the same requirements as DNS infrastructure for stability, redundancy, and network performance. This means XRI authority and proxy resolution services are subject to the same requirements as DNS nameservers. For example:
  - Critical authority or proxy resolution servers SHOULD be operated from a minimum of two
    physically separate network locations to prevent a single point of failure.
- Authority or proxy resolution servers handling heavy loads SHOULD operate from multiple
   servers and take advantage of load balancing technologies.
- However, such capabilities are effective only if resolvers or other client applications implement proper failover handling. Because XRI resolution takes place at a layer above DNS resolution, resolvers have two ways to discover additional network endpoints at which authority or proxy resolution services are available.
- DNS round robin/failover. The domain name of an authority resolution service endpoint URI may be associated with more than one IP address.
  - XRI round robin/failover. The XRDS document describing an XRI authority may publish
    multiple URI elements for its authority resolution service endpoint, or multiple authority
    resolution service endpoints, or both.

- 1356 To take advantage of both these options, the following rules apply to failover handling:
- A resolver SHOULD first try an alternate IP address for the current authority resolution service endpoint if the endpoint uses DNS round robin.
  - 2. If all alternate IP addresses fail, a resolver MUST try the next highest priority authority resolution URI in the current authority resolution service endpoint, if available.
    - If all URIs in the current authority resolution service endpoint fail, a resolver MUST try the next highest priority authority resolution service endpoint, if available, until all authority resolution service endpoints are exhausted.
    - 4. A resolver SHOULD only return an error if all network endpoints associated with the authority resolution service fail to respond.
- 1366 IMPORTANT: These rules also apply to any client of an XRI proxy resolver. Failure to observe this warning means the proxy resolver can become a point of failure.
- 1368 One final consideration: DNS caching mechanisms should respect the TTL (Time To Live)
- 1369 settings in DNS records. However, different software languages and frameworks handle DNS
- 1370 caching differently. It is RECOMMENDED to check the default settings to ensure that a library or
- application is not caching DNS results indefinitely.

1359

1360 1361

1362

1363

1364

1365

1372

## 9.1.5 Community Root Authorities

- 1373 Identifier management policies are defined on a community-by-community basis. For XRI
- 1374 identifier authorities, the resolution community is specified by the first (leftmost) subsegment of
- the authority component of the XRI. This is referred to as the *community root authority*, and it
- represents the authority server(s) that answer resolution queries at this root. When a resolution
- 1377 community chooses to create a new community root authority, it SHOULD define policies for
- 1378 assigning and managing identifiers under this authority. Furthermore, it SHOULD define what
- resolution protocol(s) may be used for these identifiers.
- 1380 For an XRI authority, the community root may be either a global context symbol (GCS) character
- or top-level cross-reference as specified in section 2.2.1.1 of [XRISyntax]. In either case, the
- 1382 corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution
- 1383 service endpoints for that community.
- 1384 The community root authority SHOULD publish a self-describing XRDS document as defined in
- 1385 section 9.1.6. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as
- the community's root authority resolution service endpoints. This community root XRDS
- 1387 document, or its location, must be known a priori and is part of the configuration of an XRI
- 1388 resolver, similar to the specification of root DNS servers for a DNS resolver. Note that it is not
- 1389 strictly necessary to publish this information in an XRDS document—it may be supplied in any
- format that enables configuration of the XRI resolvers in the community. However, publishing a
- 1391 self-describing XRDS document at a known location simplifies this process and enables dynamic
- 1392 configuration of community resolvers.
- 1393 As a best practice, it is RECOMMENDED that community root XRDS document contain:
- The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML trusted resolution is supported.
- Both of the above if HTTPS+SAML trusted resolution is supported.
- The service endpoints and supported media types of the community's XRI proxy resolver(s) if proxy resolution is supported.
- For a list of public community root authorities and the locations of their community root XRDS documents, see the Wikipedia entry on XRI **[WikipediaXRI]**.

## 9.1.6 Self-Describing XRDS Documents

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

- 1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution service endpoint, it may use the resolution protocol specified in section 6 to request an XRDS document directly from this HTTP(S) URI. This HTTP(S) URI may be known a priori (as is often the case with community root authorities, above), or it may be discovered from other identifier authorities via the resolution protocols defined in this specification.
- 2. If the resolver knows: a) an XRI of the target authority as a community root authority, and b) an HTTP(S) URI for a proxy resolver configured for this community root authority, it may use the proxy resolution protocol specifed in section 11 to query the proxy resolver for the community root authority XRI. This query MUST include only a single subsegment identifying the community root authority and MUST NOT include any additional subsegments.
- 1418 If an identifier authority had an authority resolution service endpoint at
- 1419 http://example.com/auth-res-service/, an example of the first method would be to
- issue an HTTP(S) GET request to that URI with an Accept header specifying the content type
- 1421 application/xrds+xml. See section 6.3 for more details.
- 1422 If an identifier authority with the community root authority identifier xri://(example) was
- registered with the XRI proxy resolver http://xri.example.com/, an example of the second
- method would be to issue an HTTP(S) GET request to the following URI:
- 1425 http://xri.example.com/(example)?\_xrd\_r=application/xrds+xml
- Note that a proxy resolver may use the first method to publish its own self-describing XRDS
- document at the HTTP(S) URI(s) for its proxy resolution service.
- 1428 IMPORTANT: A self-describing XRDS document MUST only be issued by an identifier authority
- when describing itself. It MUST NOT be included in an XRDS document when describing a
- 1430 different identifier authority. In the latter case the self-describing XRDS document for the
- 1431 community root authority is implicit.

## 9.1.7 Qualified Subsegments

- 1433 A qualified subsegment is defined by the productions whose names start with xri-subseg in
- section 2.2.3 of [XRISyntax] including the leading syntactic delimiter ("\*" or "!"). A qualified
- 1435 subsegment MUST include the leading syntactic delimiter even if it was optionally omitted in the
- original XRI (see section 2.2.3 of [XRISyntax]).
- 1437 If the first subsegment of an XRI authority is a GCS character and the following subsegment does
- 1438 not begin with a "\*" (indicating a reassignable subsegment) or a "!" (indicating a persistent
- 1439 subsegment), then a "\*" is implied and MUST be added when constructing the qualified
- 1440 subsegment as specified in section 9.1.7. Table 12 and Table 13 illustrate the differences
- between parsing a reassignable subsegment following a GCS character and parsing a cross-
- 1442 reference, respectively.

1443

1432

14021403

1404

1405

1406

1407

1408 1409

1410

1411

1412

1413

1414 1415

1416

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

1445 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

1446 Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.

### 9.1.8 Cross-References

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

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

Example XRI	Next Authority URI after resolving xri://@!a!b
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)

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

## 9.1.9 Selection of the Next Authority Resolution Service Endpoint

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

#### 9.1.10 Construction of the Next Authority URI 1466

- 1467 Once the next authority resolution service endpoint is selected, the resolver MUST construct a
- 1468 URI for the next HTTP(S) request, called the Next Authority URI, by concatenating two strings as
- 1469 specified in this section.

1472

1473 1474

1475

- 1470 The first string is called the Next Authority Resolution Service Endpoint URI. To construct it, the resolver MUST: 1471
  - 1. Select the highest priority URI of the highest priority authority resolution service endpoint selected in section 9.1.9.
    - 2. Apply the service endpoint URI construction algorithm based the value of the append attribute as defined in section 13.7.
- 1476 Append a forward slash ("/") if the URI does not already end in a forward slash.
- 1477 The second string is called the Next Authority String and it consists of either:
- 1478 The next fully qualified subsegment to be resolved (see section 9.1.7), or
- 1479 In the case of recursing resolution, the next fully qualified subsegment to be resolved plus 1480 any additional subsegments for which recursing resolution is requested (see section 9.1.11).
- 1481 The final step is to append the Next Authority String directly to the Next Authority Resolution
- Service Endpoint URI. The resulting URI is called the Next Authority URI. 1482
- BACKWARDS COMPATIBILITY NOTE: Earlier versions of this specification required the Next 1483
- 1484 Authority String to be appended to the path component of the Next Authority Resolution Service
- 1485 Endpoint URI. This rule was changed to give XRI authorities greater control over the structure of
- 1486 incoming resolution requests—for example, to enable Next Authority Strings to appear as query
- 1487 parameters.

1497

- 1488 Construction of the Next Authority URI is more formally described in this pseudocode for
- 1489 resolving a "next-auth-string" via a "next-auth-res-sep-uri":

```
1490
           if (next-auth-res-sep-uri does not end in "/"):
1491
               append "/" to next-auth-res-sep-uri
1492
1493
           if (next-auth-string is not preceded with "*" or "!" delimiter):
1494
               prepend "*" to next-auth-string
1495
1496
           append uri-escape(next-auth-string) to next-auth-res-sep-uri
```

## 9.1.11 Recursing Authority Resolution

- 1498 If an authority server offers recursing resolution, an XRI resolver MAY request resolution of
- 1499 multiple authority subsegments in one transaction. If a resolver makes such a request, the
- 1500 responding authority server MAY perform the additional recursing resolution steps requested. In
- 1501 this case the recursing authority server acts as a resolver to the other authority resolution service 1502 endpoints that need to be queried. Alternatively, the recursing authority server may retrieve XRDs
- from its local cache until it reaches a subsegment whose XRD is not locally cached, or it may 1503
- 1504 simply recurse only as far as it is authoritative.
- 1505 If an authority server performs any recursing resolution, it MUST return an ordered list of
- 1506 xrd:XRD elements (and nested xrd:XRDS elements if Redirects or Refs are followed as
- 1507 specified in section 12) in an xrd: XRDS document for all subsegments resolved as defined in
- 1508 section 8.2.1.
- 1509 A recursing authority server MAY resolve fewer subsegments than requested by the resolver. The
- 1510 recursing authority server is under no obligation to resolve more than the first subsegment (for
- which it is, by definition, authoritative). 1511

- 1512 If the recursing authority server does not resolve the entire set of subsegments requested, the
- 1513 resolver MUST continue the authority resolution process itself. At any stage, however, the
- 1514 resolver MAY request recursing resolution of any or all of the remaining authority subsegments.

## 9.2 IRI Authority Resolution

- 1516 From the standpoint of generic authority resolution, an IRI authority component represents either
- 1517 a DNS name or an IP address at which an XRDS document describing the authority may be
- 1518 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET
- request to a URI constructed from the IRI authority component. The resulting XRDS document
- 1520 can then be consumed in the same manner as one obtained using XRI authority resolution.
- 1521 While the use of IRI authorities provides backwards compatibility with the large installed base of
- 1522 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of
- abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities
- are NOT RECOMMENDED for new deployments of XRI identifiers.
- 1525 This section defines IRI authority resolution as a simple extension to the XRI authority resolution
- 1526 protocol defined in the preceding section.

## 1527 **9.2.1 Service Type and Media Type**

- 1528 Because IRI authority resolution takes place at a level "below" XRI authority resolution, it cannot
- be described in an XRD, and thus there is no corresponding resolution service type. IRI authority
- resolution uses the same media type as generic XRI authority resolution.

#### 1531 **9.2.2 Protocol**

1515

1534

1535

1536 1537

1538

1539

1540

1541

1543

1544

1545

1546

1549

- Following are the normative requirements for IRI authority resolution that differ from generic XRI authority resolution:
  - 1. The Next Authority URI (section 9.1.10) is constructed by extracting the entire IRI authority component and prepending the string http://. See the exception in section 9.2.3.
    - 2. The HTTP GET request MUST include an HTTP Accept header containing only the following:

Accept: application/xrds+xml

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

1542 Host: example.com

- 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document containing the XRD describing that authority.
  - 5. The responding server MUST use the value of the Host: header to populate the xrd:XRD/xrd:Query element in the resulting XRD.
- Note that because IRI authority resolution is required to process the entire IRI authority component in a single step, recursing authority resolution does not apply.

## 9.2.3 Optional Use of HTTPS

- 1550 Section 10 of this specification defines trusted resolution only for XRI authorities. Trusted
- resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to
- 1552 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY
- 1553 use HTTPS as the access protocol for retrieving the authority's XRD. If the resolver is satisfied,

via transport level security mechanisms, that the response is from the expected IRI authority, the resolver MAY consider this an HTTPS trusted resolution response as defined in section 10.1.

# 10 Trusted Authority Resolution Service

- 1557 This section defines three options for performing trusted XRI authority resolution as an extension
- 1558 of the generic authority resolution protocol defined in section 9.1—one using HTTPS, one using
- 1559 SAML assertions, and one using both.

#### 1560 **10.1 HTTPS**

1556

1565

1573

1574

1575

1576 1577

1578 1579

1580

1581

1582

1583 1584

1585

1586

- 1561 HTTPS authority resolution is a simple extension to generic authority resolution in which all
- 1562 communication with authority resolution service endpoints is carried out over HTTPS. This
- 1563 provides transport-level security and server authentication, however it does not provide message-
- level security or a means for a responder to provide different responses for different requestors.

## 10.1.1 Service Type and Service Media Type

1566 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

- 1567 Table 15: Service Type and Service Media Type values for HTTPS trusted authority resolution.
- 1568 An HTTPS trusted resolution service endpoint advertised in an XRDS document MUST use the
- 1569 Service Type identifier and Service Media Type identifier (including the https=true parameter)
- defined in Table 15. In addition, the identifier authority MUST use an HTTPS URI as the value of
- 1571 the xrd: URI element(s) for this service endpoint.

#### 1572 **10.1.2 Protocol**

- Following are the normative requirements for HTTPS trusted authority resolution that differ from generic authority resolution (section 9.1):
  - All authority resolution service endpoints MUST be selected using the values defined in Table 15.
  - 2. All authority resolution requests, including the starting request to a community root authority, MUST use the HTTPS protocol as defined in [RFC2818]. This includes all intermediate redirects, as well as all authority resolution requests resulting from Redirect and Ref processing as defined in section 12. A successful HTTPS response MUST be received from each authority in the resolution chain or the output MUST be error.
  - 3. All authority resolution requests MUST contain an HTTPS Accept header with the media type identifier defined in Table 15 (including the https=true subparameter).
  - 4. If the resolver finds that an authority in the resolution chain does not support HTTPS at any of its authority resolution service endpoints, the resolver MUST return a 23x error as defined in section 15.

## 1587 **10.2 SAML**

In SAML trusted resolution, the resolver uses the Resolution Output Format subparameter

saml=true and the authority server responds with an XRDS document containing an XRD with

an additional element—a digitally signed SAML [SAML] assertion that asserts the validity of the

containing XRD. SAML trusted resolution provides message integrity but does not provide

confidentiality. For this reason is RECOMMENDED to combine SAML trusted resolution with

- 1593 HTTPS trusted resolution as defined in section 10.3. Message confidentiality may also be
- 1594 achieved with other security protocols used in conjunction with this specification. SAML trusted
- resolution also does not provide a means for an authority to provide different responses for
- 1596 different requestors; client authentication is explicitly out-of-scope for version 2.0 of XRI
- 1597 resolution.

1598

1606

1609

1610

1611 1612

1613

1614

1615

1616

1617

1626

1627 1628

1629

## 10.2.1 Service Type and Service Media Type

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

- 1600 Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.
- 1601 A SAML trusted resolution service endpoint advertised in an XRDS document MUST use the
- 1602 Service Type identifier and Service Media Type identifier defined in Table 16 (including the
- 1603 saml=true subparameter). In addition, for transport security the identifier authority SHOULD
- 1604 offer at least one HTTPS URI as the value of the xrd:URI element(s) for this service endpoint.

#### 1605 **10.2.2 Protocol**

#### 10.2.2.1 Client Requirements

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

- All authority resolution service endpoints MUST be selected using the values defined in Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an authority unless the authority advertises a resolution service endpoint matching these values.
- 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is RECOMMENDED for confidentiality.
- 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the media type identifier defined in Table 16 (including the saml=true subparameter). This is the media type of the requested response.
- 1618 IMPORTANT: Clients willing to accept either generic or trusted responses MAY use a combination of media type identifiers in the Accept header as described in section 14.1 of
- 1620 [RFC2616]. Media type identifiers SHOULD be ordered according to the client's preference for
- the media type of the response. If a client performing generic authority resolution receives an
- 1622 XRD containing SAML elements, it MAY choose not to validate the signature or perform any
- 1623 processing of these elements.
- 4. A resolver MAY request recursing authority resolution of multiple subsegments as defined in section 10.2.3.
  - 5. The resolver MUST individually validate each XRD it receives in the resolution chain according to the rules defined in section 10.2.4. When xrd:XRD elements come both from freshly-retrieved XRDS documents and from a local cache, a resolver MUST ensure that these requirements are satisfied each time a resolution request is performed.

#### 10.2.2.2 Server Requirements

- For an authority server, trusted resolution is identical to the generic resolution protocol (section 9.1) with the addition of the following requirements:
  - 1. The HTTP(S) response to a trusted resolution request MUST include a content type of application/xrds+xml;saml=true.
  - 2. The XRDS document returned by the resolution service MUST contain a saml:Assertion element as an immediate child of the xrd:XRD element that is valid per the processing rules described by [SAML].
  - 3. The saml:Assertion element MUST contain a valid enveloped digital signature as defined by [XMLDSig] and as constrained by section 5.4 of [SAML].
  - 4. The signature MUST apply to the xrd:XRD element that contains the signed SAML assertion. Specifically, the signature MUST contain a single ds:SignedInfo/ds:Reference element, and the URI attribute of this reference MUST refer to the xrd:XRD element that is the immediate parent of the signed SAML assertion. The URI reference MUST NOT be empty and it MUST refer to the identifier contained in the xrd:XRD/@xml:id attribute.
  - 5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain a ds:KeyInfo element. If this element is included, it MUST describe the key used to verify the digital signature element. However, because the signing key is known in advance by the resolution client, the ds:KeyInfo element SHOULD be omitted from the ds:Signature element of the SAML assertion.
  - 6. The xrd: XRD/xrd: Query element MUST be present, and the value of this field MUST match the XRI authority subsegment requested by the client.
  - 7. The xrd:XRD/xrd:ProviderID element MUST be present and its value MUST match the value of the xrd:XRD/xrd:Service/xrd:ProviderID element in an XRD advertising availability of trusted resolution service from this authority as required in section 10.2.5.
  - 8. The xrd:XRD/saml:Assertion/saml:Subject/saml:NameID element MUST be present and equal to the xrd:XRD/xrd:Ouery element.
  - 9. The NameQualifier attribute of the xrd:XRD/saml:Assertion/saml:Subject/saml:NameID element MUST be present and MUST be equal to the xrd:XRD/xrd:ProviderID element.
  - 10. There MUST be exactly one saml:AttributeStatement present in the xrd:XRD/saml:Assertion element. It MUST contain exactly one saml:Attribute element with a Name attribute value of xri://\$xrd\*(\$v\*2.0). This saml:Attribute element MUST contain exactly one saml:AttributeValue element whose text value is a URI reference to the xml:id attribute of the xrd:XRD element that is the immediate parent of the saml:Assertion element.

## **10.2.3 Recursing Authority Resolution**

1669 If a resolver requests trusted resolution of multiple authority subsegments (see section 9.1.8), a
1670 recursing authority server SHOULD attempt to perform trusted resolution on behalf of the resolver
1671 as described in this section. However, if the resolution service is not able to obtain trusted XRDs
1672 for one or more additional recursing subsegments, it SHOULD return only the trusted XRDs it has
1673 obtained and allow the resolver to continue.

## 1674 10.2.4 Client Validation of XRDs

1677

1678

1679

1680

1681

1682

1683

1684

1685 1686

1687

1688

1689

1690

1691 1692

1693

1694

1695

1696

1697

1698

1699

1700

1701

1702

1703

1704

- For each XRD returned as part of a trusted resolution request, the resolver MUST validate the XRD according to the rules defined in this section.
  - 1. The xrd:XRD/saml:Assertion element MUST be present.
  - 2. This assertion MUST be valid per the processing rules described by [SAML].
  - 3. The saml: Assertion MUST contain a valid enveloped digital signature as defined by [XMLDSig] and constrained by Section 5.4 of [SAML].
    - 4. The signature MUST apply to the xrd:XRD element containing the signed SAML assertion. Specifically, the signature MUST contain a single ds:SignedInfo/ds:Reference element, and the URI attribute of this reference MUST refer to the xml:id attribute of the xrd:XRD element that is the immediate parent of the signed SAML assertion.
    - 5. If the digital signature enveloped by the SAML assertion contains a ds:KeyInfo element, the resolver MAY reject the signature if this key does not match the signer's expected key as specified by the ds:KeyInfo element present in the XRD Descriptor that was used to describe the current authority. See section 10.2.5.
    - 6. The value of the xrd:XRD/xrd:Query element MUST match the subsegment whose resolution resulted in the current XRD.
    - 7. The value of the xrd:XRD/xrd:ProviderID element MUST match the value of the xrd:XRD/xrd:ProviderID element in any XRD advertising availability of trusted resolution service from this authority as required in section 10.2.5.
    - 8. The value of the xrd:XRD/xrd:ProviderID element MUST match the value of the NameQualifier attribute of the xrd:XRD/saml:Assertion/saml:Subject/saml:NameID element.
    - 9. The value of the xrd:XRD/xrd:Query element MUST match the value of the xrd:XRD/saml:Assertion/saml:Subject/saml:NameID element.
  - 10. There MUST exist exactly one xrd:XRD/saml:Assertion/saml:AttributeStatment with exactly one saml:Attribute element that has a Name attribute value of xri://\$xrd\*(\$v\*2.0). This saml:Attribute element must have exactly one saml:AttributeValue element whose text value is a URI reference to the xml:id attribute of the xrd:XRD element that is the immediate parent of the signed SAML assertion.
- 1706 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result 1707 MUST NOT be considered a valid trusted resolution response as defined by this specification.
- 1708 Note that this does not preclude a resolver from considering alternative resolution paths. For
- 1709 example, if an XRD advertising SAML trusted resolution service has two or more
- 1710 xrd:XRD/xrd:Service/xrd:URI elements and the response from one service endpoint fails
- 1711 to meet the requirements above, the client MAY repeat the validation process using the second
- 1712 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as
- 1713 defined by this document and SAML trusted resolution may continue.
- 1714 If the above requirements are met, and the code attribute of the xrd:XRD/xrd:ServerStatus
- 1715 element is 100 (SUCCESS), the resolver MUST add an xrd:XRD/xrd:Status element
- 1716 reporting a status of 100 (SUCCESS) as specified in section 15. Note that this added element
- 1717 MUST be disregarded if a consuming application wishes to verify the SAML signature itself. (If
- 1718 necessary, the consuming application may request the XRDS document it wishes to verify directly
- 1719 from the SAML authority resolution server.)
- 1720 If all SAML trusted resolution paths fail, the resolver MUST return the appropriate 23x trusted
- 1721 resolution error as defined in section 15.

## 1722 10.2.5 Correlation of ProviderID and KeyInfo Elements

- 1723 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at
- 1724 least one unique persistent identifier expressed in the
- 1725 xrd:XRD/xrd:Service/xrd:ProviderID element of any XRD advertising trusted authority
- 1726 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI
- 1727 authority. While a ProviderID may be any valid URI that meets these requirements, it is
- 1728 STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI
- 1729 [XRISyntax] or a URN [RFC2141].
- 1730 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in
- 1731 an XRD advertising SAML trusted authority resolution service with the response received from a
- 1732 SAML trusted resolution service endpoint. If the signed XRD response contains the same
- 1733 ProviderID as the XRD used to advertise a service, and the resolver has reason to trust the
- signature, the resolver can trust that the XRD response has not been maliciously replaced with
- 1735 another XRD.
- 1736 There is no defined discovery process for the ProviderID for a community root authority; it must
- 1737 be published in a self-describing XRDS document (or other equivalent description—see sections
- 1738 9.1.5 and 9.1.6) and verified independently. Once the community root XRDS document is known,
- the ProviderID for delegated XRI authorities within this community MAY be discovered using the
- 1740 xrd:XRD/xrd:Service/xrd:ProviderID element of authority resolution service endpoints.
- 1741 This trust mechanism MAY also be used for other services offered by an authority.
- 1742 In addition, the metadata necessary for SAML trusted authority resolution or other SAML [SAML]
- 1743 interactions MAY be discovered using the ds:KeyInfo element (section 4.2.) Again, if this
- 1744 element is present in an XRD advertising SAML authority resolution service (or any other
- 1745 service), and the client has reason to trust this XRD, the client MAY use the associated
- 1746 ProviderID to correlate the contents of this element with a signed response.
- 1747 To assist resolvers in using this key discovery mechanism, it is important that trusted authority
- 1748 servers be configured to sign responses in such a way that the signature can be verified using the
- 1749 correlated ds: KeyInfo element. For more information, see [SAML].

#### 1750 **10.3 HTTPS+SAML**

#### 10.3.1 Service Type and Service Media Type

1752 The protocol defined in this section is identified by the values in Table 17.

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

- 1753 Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.
- 1754 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST
- use the Service Type identifier and Service Media Type identifier defined in Table 17 (including
- 1756 the https=true and saml=true subparameters). In addition, the identifier authority MUST use
- an HTTPS URI as the value of the xrd: URI element(s) for this service endpoint.

1758

### 1759 **10.3.2 Protocol**

1761

1762

1763

1764 1765

1766

1767

1768

1769

1770 1771

1772

1760 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- All authority resolution service endpoints MUST be selected using the values defined in Table 17.
- 2. All authority resolution requests and responses, including the starting request to a community root authority, MUST conform to both the requirements of the HTTPS trusted resolution protocol defined in section 10.1 and the SAML trusted resolution protocol defined in section 10.2.
- 3. All authority resolution requests MUST contain an HTTPS Accept header with the media type identifier defined in Table 17 (including both the https=true and saml=true parameters). This MUST be interpreted as the value of the Resolution Output Format input parameter.
- 4. If the resolver finds that an authority in the resolution chain does not support both HTTPS 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
- 1778 query parameters in the HXRI.

1773

1785

1786

1787

1788

1796

1797

1798

1806

- 1779 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 QXRIs 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

1789 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

- 1790 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.
- 1799 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
- decented. However, advertising a proxy resolution service in the Artise decentric for a
- 1802 community root authority (sections 9.1.3 and 9.1.6) can be very useful for applications that need
- 1803 to consume XRI proxy resolution services or automatically generate HXRIs for resolution by non-
- 1804 XRI-aware clients in that community. Those applications may discover the current URI(s) and
- 1805 resolution capabilities of a proxy resolver from this source.

#### **11.2 HXRIs**

- 1807 The first step in an HTTP binding of the XRI resolution interface is to specify how the QXRI
- 1808 parameter is passed within an HTTP(S) URI. Besides providing a binding for proxy resolution,
- 1809 defining a standard syntax for expressing an XRI as an HTTP XRI (HXRI) has two other benefits:

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

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

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

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

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

## 11.3 HXRI Query Parameters

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

Logical Parameter Name	HXRI Component	HXRI Query Parameter Name
QXRI	Entire path and query string of HXRI (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

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

- Following are the rules for the use of the parameters specified in Table 19.
- 1843 1. The QXRI MUST be normalized as specified in section 11.2.
- If the original QXRI has an existing query component, the HXRI query parameters MUST
   be appended to that query component.

1846 IMPORTANT: The query parameter names in Table 19 were chosen to minimize the probability of collision with any pre-existing query parameter names in the QXRI. If there is any conflict, the pre-existing query parameter names MUST be percent-encoded prior to transformation into an HXRI.

- After proxy resolution, the HXRI query parameters MUST subsequently be removed from the QXRI query component. The existing QXRI query component MUST NOT be altered in any other way, i.e., it must be passed through with no changes in parameter order, escape encoding, etc.
- If the original QXRI does not have a query component, one MUST be added to pass any HXRI query parameters. After proxy resolution, this query component MUST be entirely removed.
- 5. If the original QXRI had a null query component (only a leading question mark), or a query component consisting of only question marks, one additional leading question mark MUST be added before adding any HXRI query parameters. After proxy resolution, any HXRI query parameters and exactly one leading question mark MUST be removed. See the URI construction steps defined in section 13.6.
- 6. Each HXRI query parameter MUST be delimited from other parameters by an ampersand ("&").
- 7. Each HXRI query parameter MUST be delimited from its value by an equals sign ("=").
- 8. If an HXRI query parameter includes one of the media type parameters defined in Table 6, it MUST be delimited from the HXRI query parameter with a semicolon (";").
- 9. The fully-composed HXRI MUST be encoded and decoded as specified in section 11.4.
- 10. If any HXRI query parameter name is included but its value is empty, the value of the parameter MUST be considered null.

## 11.4 HXRI Encoding/Decoding Rules

- 1871 To conform with the typical requirements of web server URI parsing libraries, HXRIs MUST be
- 1872 encoded prior to input to a proxy resolver and decoded prior to output from a proxy resolver.
- 1873 Because web server libraries typically perform some of these decoding functions automatically,
- 1874 implementers MUST ensure that a proxy resolver, when used in conjunction with a specific web
- 1875 server, accomplishes the full set of HXRI decoding steps specified in this section. In particular,
- 1876 these decoding steps MUST be performed prior to any comparison operations defined in this
- 1877 specification.

1850

1851

1852

1853

1854

1855

1856

1857

1858

1859

1860

1861

1862

1863

1864 1865

1866

1867

1868

1869

- 1878 Before any HXRI-specific encoding steps are performed, the QXRI portion of the HXRI (including
- 1879 all HXRI query parameters) MUST be transformed into URI-normal form as defined in section 2.3
- 1880 of [XRISyntax]. This means characters not allowed in URIs, such as SPACE, or characters that
- 1881 are valid only in IRIs, such as UCS characters above the ASCII range, MUST be percent
- encoded. Also, the plus sign character ("+") MUST NOT be used to encode the SPACE character
- because in decoding the percent-encoded sequence %2B MUST be interpreted as the plus sign
- 1884 character ("+").
- 1885 Once the HXRI is in URI-normal form, the following sequence of encoding steps MUST be
- 1886 performed in the order specified before an HXRI is submitted to a proxy resolver.
- 1887 IMPORTANT: this sequence of steps is not idempotent, so it MUST be performed only once.

- 1888 1889 1890
- 1. First, in order to preserve percent-encoding when the HXRI is passed through a web server, all percent signs MUST be themselves percent-encoded, i.e., a SPACE encoded as \$20 will become \$2520.
- 1891 1892 1893
- 2. Second, to prevent misinterpretation of HXRI query parameters, any occurrences of the ampersand character ("&") within an HXRI query parameter that are NOT used to delimit it from another query parameter MUST be percent encoded using the sequence \$26.
- 1894 1895 1896
- 3. Third, to prevent misinterpretation of the semicolon character by the web server, any semicolon used to delimit one of the media type parameters defined in Table 6 from the media type value MUST be percent-encoded using the sequence \$3B.
- 1897 1898 1899
- To decode an encoded HXRI back into URI-normal form, the above sequence of steps MUST be performed in reverse order. Again, the sequence is not idempotent so it MUST be performed only once.
- 1900 1901

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

  QXRI https://xri.example.com/=example\*résumé/path?query
  \_xrd\_r \_\_xrd\_r=application/xrds+xml;https=true;sep=true
  - \_xrd\_r \_\_xrd\_r=application/xrds+xml;https=true;sep=true

    \_xrd\_t \_\_xrd\_t=http://example.org/test?a=1&b=hello planete

    \_xrd\_m \_\_xrd\_m=application/atom+xml
- 1904
- Table 20: Example of HXRI components prior to transformation to URI-normal form.
- 1905
- Table 21 illustrates these components after transformation to URI-normal form. Characters that have been percent-encoded are in **blue**. Characters still requiring percent encoding according to the rules defined in this section are highlighted in **red**.

1906	
1907	

- 1908
- Table 21: Example of HXRI components after transformation to URI-normal form.
- 1909
- Table 22 illustrates the components after all encoding rules defined in this section are applied.

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

- 1910
- Table 22: Example of HXRI components after application of the required encoding rules.

## 1912 Following is the fully-encoded HXRI:

19231924

1925

1926

1927

1928

1929

1930

1931

1932

1933

1934

1935 1936

1937

1947

```
1913 https://xri.example.com/=example*r%25E9sum%25E9/path?query
1914 &_xrd_r=application/xrds+xml%3Bhttps=true%3Bsep=true
1915 &_xrd_t=http://example.org/test?a=1%26b=hello%2520plan%25E8te
1916 &_xrd_m=application/atom+xml
```

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

```
1919 https://xri.example.com/=example*r%E9sum%E9/path?query
1920 &_xrd_r=application/xrds+xml;https=true;sep=true
1921 &_xrd_t=http://example.org/test?a=1&b=hello%20plan%E8te
1922 &_xrd_m=application/atom+xml
```

## 11.5 HTTP(S) Accept Headers

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

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

## 11.6 Null Resolution Output Format

- 1938 Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a
- resolution request where the Resolution Output Format input parameter value is null—either
- because this parameter is absent or because it was explicitly set to null using the <code>\_xrd\_r</code> query parameter.
- 1041 parameter.
- 1942 If the value of the Resolution Output Format value is null, a resolver MUST proceed as if the
- 1943 following media type parameters had the following values: https=false, saml=false,
- 1944 refs=true, sep=true, nodefault t=false, nodefault p=false,
- 1945 nodefault m=false, and uric=false. In addition, the output MUST be an HTTP(S) redirect
- 1946 as defined in the following section.

## 11.7 Outputs and HTTP(S) Redirects

- 1948 For all values of the Resolution Output Format parameter except null, a proxy resolver MUST
- 1949 follow the output rules defined in section 8.2.
- 1950 If the value of the Resolution Output Format is null, and the output is not an error, a proxy
- 1951 resolver MUST follow the rules for output of a URI List as defined in section 8.2.3. However,
- 1952 instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as
- an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service
- 1954 Media Type parameter.
- 1955 If the output is an error, a proxy resolver SHOULD return a human-readable error message as
- 1956 specified in section 15.4.

1957 1958 1959 1960	These rules enable XRI proxy resolvers to serve clients that do not understand XRI syntax or resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept header (if any).

## 11.8 Differences Between Proxy Resolution Servers

1961

1968

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

## 11.9 Combining Authority and Proxy Resolution Servers

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

# 12Redirect and Ref Processing

 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:

- 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.
- 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.

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 append attribute as the xrd:URI 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	redirect	ref
Number of XRDs in nested XRDS document	1	1 or more

Table 23: Comparison of Redirect and Ref elements.

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

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*.

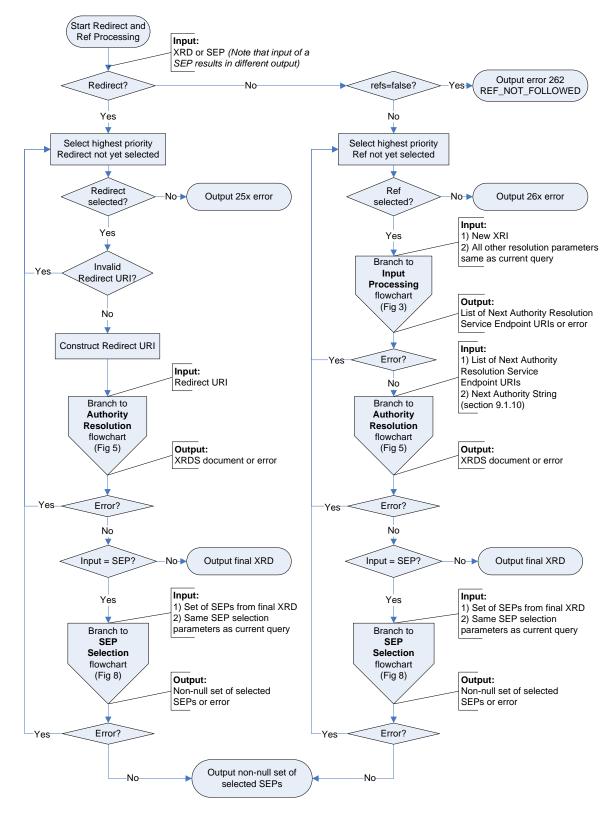


Figure 7: Redirect and Ref processing flowchart.

# 12.1 Cardinality

2002

2008

2009

2010

2012

2013

2014

20152016

2017

2018

2019

2020

2021

2022

2023

2024

2025

2026

2027

2028

2029

2030

2031

2032

2033

2034

2035

2036

2037

2038

20392040

2041

- 2003 Redirect and Ref elements may be used both at the XRD level (as a child of the xrd:XRD 2004 element) and the SEP level (as a child of the xrd:XRD/xrd:Service element) within an XRD.

  2005 In both cases, to simplify processing, the XRD schema (Appendix B) enforces the following rules:
- At the XRD level, an XRD MAY contain only one of the following: zero-or-more 2007 xrd:Redirect or zero-or-more xrd:Ref elements.
  - At the SEP level, a SEP MAY contain only one of the following: zero-or-more xrd:URI elements, zero-or-more xrd:Redirect elements, or zero-or-more xrd:Ref elements.

### 12.2 Precedence

2011 XRDS authors should take special note of the following precedence rules for Redirect and Refs.

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

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

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

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

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

# 12.3 Redirect Processing

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

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

- 1. To process a Redirect at either the XRD or SEP level, the resolver MUST begin by selecting the highest priority xrd:XRD/xrd:Redirect element in the XRD or SEP.
- 2. If the value of the resolution subparameter https is FALSE, or the subparameter is absent or empty, the value of the selected xrd:Redirect element MUST be EITHER a valid HTTP URI or a valid HTTPS URI. If not, the resolver MUST select the next highest priority xrd:Redirect element. If all instances of this element fail, the resolver MUST stop and return the error 251 INVALID\_REDIRECT in the XRD containing the Redirect or as a plain text error message as specified in section 15.
- 3. If the value of the resolution subparameter https is TRUE, the value of the selected xrd:Redirect element MUST be a valid HTTPS URI. If not, the resolver MUST select the next highest priority xrd:Redirect element. If all instances of this element fail, the resolver MUST stop and return the error 252 INVALID\_HTTPS\_REDIRECT in the XRD containing the Redirect or as a plain text error message as specified in section 15.
- 4. Once a valid xrd:Redirect element has been selected, if the xrd:XRD/xrd:Redirect element includes the append attribute, the resolver MUST construct the final HTTP(S) URI as defined in section 13.7.
- 5. The resolver MUST request a new XRDS document from the final HTTP(S) URI using the protocol defined in section 9.1.3. If the Resolution Output Format is an XRDS document, the resolver MUST embed a nested XRDS document containing an XRD representing the Redirect as specified in section 12.5.
- 6. If resolution of an xrd:Redirect element fails during the authority resolution phase of the original resolution query, or if resolution of an xrd:Redirect element fails during the optional service endpoint selection phase OR the final XRD does not contain the requested SEP, then the resolver MUST report the error in the final XRD of the nested XRDS document using the status codes defined in section 15. (One nested XRDS document will be added for each Redirect attempted by the resolver.) The resolver MUST then select the next highest priority xrd:Redirect element from the original XRD or SEP and repeat rule 7. For more details, see section 12.6, Recursion and Backtracking.
- 7. If resolution of all xrd:Redirect elements in the XRD or SEP that originally triggered Redirect processing fails, the resolver MUST stop and return a 25x error in the XRD containing the Redirect or as a plain text error message as specified in section 15. The resolver MUST NOT try any other SEPs even if multiple SEPs were selected as specified in section 13.
- If resolution succeeds, the resolver MUST verify the synonym elements in the new XRD
  as specified in section 14.1. If synonym verification fails, the resolver MUST stop and
  return the error specified in that section.
- If the value of the resolution subparameter saml is TRUE, the resolver MUST verify the signature on the XRD as specified in section 10.2.4. If signature verification fails, the resolver MUST stop and return the error specified in that section.
- If Redirect resolution succeeds, further authority resolution or service endpoint selection MUST continue based on the new XRD.

# 12.4 Ref Processing

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

2102 Following are the normative rules for processing of the xrd:Ref element.

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

### 12.5 Nested XRDS Documents

2144 Processing of a Redirect or Ref ALWAYS produces a new XRDS document that describes the
2145 Redirect or Ref that was followed, even if the result was an error. If the final requested Resolution
2146 Output Format is NOT an XRDS document, this new XRDS document is only needed to obtain
2147 the metadata necessary to continue or complete resolution. However, if the final requested

2148 Resolution Output Format is an XRDS document, each XRDS document produced as a result of Redirect or Ref processing MUST be nested inside the outer XRDS document immediately

2150 following the xrd:XRD element containing the xrd:Redirect or xrd:Ref element being

2151 followed. If more than one Redirect or Ref element is resolved due to an error, the corresponding

2152 nested XRDS documents MUST be included in the same order as the Redirect or Ref elements

that were followed to produce them.

2143

2156 2157

2158

2159

2160

2161

2162 2163

2164

2165

2166

2167 2168

2169

2170

21712172

2173

2174

2175 2176

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

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

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

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

### 12.5.1 Redirect Examples

#### Example #1:

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

```
2177
           <XRDS xmlns="xri://$xrds" ref="xri://@a">
2178
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2179
                <Query>*a</Query>
2180
                <ProviderID>xri://@</ProviderID>
2181
                <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2182
                <Redirect>http://a.example.com/</Redirect>
2183
2184
              </XRD>
2185
              <XRDS redirect="http://a.example.com/">
2186
                <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2187
                   <ProviderID>xri://@</ProviderID>
2188
                   <CanonicalID>xri://@!1</CanonicalID> ; SAME AS XRDS #1 CID #1
2189
2190
                   <Service>
2191
                     <Type>http://openid.net/signon/1.0</Type>
2192
                     <URI>http://openid.example.com/</URI>
```

#### Example #2:

2197

2198

2199

2200

2201

2202

2247

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

```
2203
           <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2204
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2205
                <Query>*a</Query>
2206
                 <ProviderID>xri://@</ProviderID>
2207
                <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2208
2209
                <Service>
2210
                   <Type>xri://$res*auth*($v*2.0)</Type>
2211
                   <URI>http://a.example.com/</URI>
2212
                </Service>
2213
              </XRD>
2214
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2215
                <Query>*b</Query>
2216
                <ProviderID>xri://@!1</ProviderID>
2217
                <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2
2218
                . . .
2219
                <Service>
2220
                   <Type>xri://$res*auth*($v*2.0)</Type>
2221
                   <Redirect>http://other.example.com</Redirect>
2222
                </Service>
2223
              </XRD>
2224
              <XRDS redirect="http://other.example.com">
2225
                <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2226
                   <Query>*b</Query>
2227
                   <ProviderID>xri://@!1</ProviderID>
2228
                   <CanonicalID>xri://@!1!2</CanonicalID> ;SAME AS XRDS #1 CID #2
2229
2230
                   <Service>
2231
                     <Type>xri://$res*auth*($v*2.0)</Type>
2232
                     <URI>http://b.example.com/</URI>
2233
                   </Service>
2234
                </XRD>
2235
              </XRDS>
2236
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2237
                <Query>*c</Query>
2238
                <ProviderID>xri://@!1!2</ProviderID>
2239
                <CanonicalID>xri://@!1!2!3</CanonicalID> ;XRDS #1 CID #3
2240
2241
2242
                 ...final service endpoints described here...
2243
                </Service>
2244
              </XRD>
2245
            </XRDS>
2246
```

#### 2248 **Example #3**:

2249

2250

2251

2295

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

```
2252
           <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2253
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2254
                <Query>*a</Query>
2255
                <ProviderID>xri://@</ProviderID>
2256
                <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2257
2258
                <Service>
2259
                  <Type>xri://$res*auth*($v*2.0)</Type>
2260
                   <URI>http://a.example.com/</URI>
2261
                </Service>
2262
              </XRD>
2263
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2264
                <Query>*b</Query>
2265
                <ProviderID>xri://@!1</ProviderID>
2266
                <CanonicalID>xri://@!1!2</CanonicalID>
                                                              ;XRDS #1 CID #2
2267
2268
                <Service>
2269
                   <Type>xri://$res*auth*($v*2.0)</Type>
2270
                   <URI>http://b.example.com/</URI>
2271
                </Service>
2272
              </XRD>
2273
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2274
                <Ouery>*c</Ouery>
2275
                <ProviderID>xri://@!1!2</ProviderID>
2276
                <CanonicalID>xri://@!1!2!3</CanonicalID>
                                                              ;XRDS #1 CID #3
2277
2278
                <Service>
2279
                  <Type>http://openid.net/signon/1.0</Type>
2280
                   <Redirect>http://r.example.com/openid</Redirect>
2281
2282
2283
              <XRDS redirect="http://r.example.com/openid">
2284
                <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2285
                   <ProviderID>xri://@!1!2</ProviderID>
2286
                  <CanonicalID>xri://@!1!2!3</CanonicalID> ;SAME AS XRDS #1 CID #3
2287
2288
                   <Service>
2289
                     <Type>http://openid.net/signon/1.0</Type>
2290
                     <URI>http://openid.example.com/</URI>
2291
                   </Service>
2292
                </XRD>
2293
              </XRDS>
2294
           </XRDS>
```

#### 2296 **Example #4:**

2297

2298

2299

2300

2340

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.

```
2301
           <XRDS xmlns="xri://$xrds" ref="xri://@a*b">
2302
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2303
                <Query>*a</Query>
2304
                <ProviderID>xri://@</ProviderID>
2305
                <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2306
                <Redirect>http://a.example.com/</Redirect>
2307
2308
              </XRD>
2309
              <XRDS redirect="http://a.example.com/">
2310
                <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2311
                  <ProviderID>xri://@</ProviderID>
2312
                  <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1 CID #1
2313
                  <Redirect>http://b.example.com/</Redirect>
2314
2315
                </XRD>
2316
                <XRDS redirect="http://b.example.com/">
2317
                   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2318
                     <ProviderID>xri://@</ProviderID>
2319
                     <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1 CID #1
2320
2321
                     <Service>
2322
                       <Type>xri://$res*auth*($v*2.0)</Type>
2323
                       <URI>http://b.example.com/</URI>
2324
                     </Service>
2325
                   </XRD>
2326
                </XRDS>
2327
              </XRDS>
2328
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2329
                <Query>*b</Query>
2330
                <ProviderID>xri://@!1</ProviderID>
2331
                <CanonicalID>xri://@!1!2</CanonicalID>
                                                              ;XRDS #1 CID #2
2332
2333
2334
                   <Type>xri://$res*auth*($v*2.0)</Type>
2335
                   <URI>http://b.example.com/</URI>
2336
                </Service>
2337
              </XRD>
2338
           </XRDS>
2339
```

### 12.5.2 Ref Examples

#### Example #1:

23412342

2343

2344

2345

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

```
2346
           <XRDS xmlns="xri://$xrds" ref="xri://@a">
2347
                 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2348
                   <Query>*a</Query>
2349
                   <ProviderID>xri://@</ProviderID>
2350
                   <CanonicalID>xri://@!1</CanonicalID>
                                                              ;XRDS #1 CID #1
2351
                   <Ref>xri://@x*y</Ref>
2352
                </XRD>
2353
                <XRDS ref="xri://@x*y">
2354
                   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2355
                     <Query>*x</Query>
2356
                     <ProviderID>xri://@</ProviderID>
2357
                     <CanonicalID>xri://@!7</CanonicalID>
                                                              ;XRDS #2 CID #1
2358
2359
                     <Service>
2360
                            <Type>xri://$res*auth*($v*2.0)</Type>
2361
                            <URI>http://x.example.com/</URI>
2362
                     </Service>
2363
                   </XRD>
2364
                   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2365
                     <Ouery>*y</Ouery>
2366
                     <ProviderID>xri://@!7</ProviderID>
2367
                     <CanonicalID>xri://@!7!8</CanonicalID> ;XRDS #2 CID #2
2368
2369
                     <Service>
2370
                            <Type>xri://$res*auth*($v*2.0)</Type>
2371
                            <URI>http://y.example.com/</URI>
2372
                     </Service>
2373
                     <Service>
2374
                            <Type>http://openid.net/signon/1.0</Type>
2375
                            <URI>http://openid.example.com/</URI>
2376
                     </Service>
2377
                   </XRD>
2378
                </XRDS>
2379
            </XRDS>
```

#### Example #2:

2380

2381

2382

2383

2384

2385

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

```
2386
           <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2387
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2388
                <Query>*a</Query>
                <ProviderID>xri://@</ProviderID>
2389
2390
                <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2391
2392
2393
                  <Type>xri://$res*auth*($v*2.0)</Type>
2394
                  <URI>http://a.example.com/</URI>
```

```
2395
                </Service>
2396
              </XRD>
2397
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2398
                <Query>*b</Query>
2399
                <ProviderID>xri://@!1</ProviderID>
2400
                <CanonicalID>xri://@!1!2</CanonicalID>
                                                             ;XRDS #1 CID #2
2401
2402
                <Service>
2403
                  <Type>xri://$res*auth*($v*2.0)</Type>
2404
                   <Ref>xri://@x*y</Ref>
2405
                </Service>
2406
              </XRD>
2407
              <XRDS ref="xri://@x*y">
2408
                <XRD xmlns="xri://xrd*(v*2.0)" version="2.0">
2409
                  <Query>*x</Query>
2410
                  <ProviderID>xri://@</ProviderID>
2411
                  <CanonicalID>xri://@!7</CanonicalID>
                                                             ;XRDS #2 CID #1
2412
2413
                  <Service>
2414
                     <Type>xri://$res*auth*($v*2.0)</Type>
2415
                     <URI>http://x.example.com/</URI>
2416
                   </Service>
2417
                </XRD>
2418
                <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2419
                  <Query>*y</Query>
2420
                  <ProviderID>xri://@!7</ProviderID>
2421
                  <CanonicalID>xri://@!7!8</CanonicalID>
                                                             ;XRDS #2 CID #2
2422
2423
                  <Service>
2424
                     <Type>xri://$res*auth*($v*2.0)</Type>
2425
                     <URI>http://y.example.com/</URI>
2426
                   </Service>
2427
                </XRD>
2428
              </XRDS>
2429
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2430
                <Query>*c</Query>
2431
                <ProviderID>xri://@!1!2</ProviderID>
2432
                <CanonicalID>xri://@!1!2!3</CanonicalID>
                                                              ;XRDS #1 CID #3 IS
2433
           CHILD OF XRDS #1 CID #2
2434
2435
                <Service>
2436
                 ...final service endpoints described here...
2437
                </Service>
2438
              </XRD>
2439
           </XRDS>
```

#### Example #3:

2440

2441

2442

2443

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

```
2444
           <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2445
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2446
                <Query>*a</Query>
2447
                <ProviderID>xri://@</ProviderID>
2448
                <CanonicalID>xri://@!1</CanonicalID>
                                                             ;XRDS #1 CID #1
2449
2450
                <Service>
2451
                  <Type>xri://$res*auth*($v*2.0)</Type>
2452
                   <URI>http://a.example.com/</URI>
2453
                </Service>
```

```
2454
              </XRD>
2455
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2456
                 <Query>*b</Query>
2457
                 <ProviderID>xri://@!1</ProviderID>
2458
                <CanonicalID>xri://@!1!2</CanonicalID>
                                                               ;XRDS #1 CID #2
2459
2460
                <Service>
2461
                   <Type>xri://$res*auth*($v*2.0)</Type>
2462
                   <URI>http://a.example.com/</URI>
2463
                 </Service>
2464
              </XRD>
2465
              <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2466
                <Query>*c</Query>
2467
                 <ProviderID>xri://@!1!2</ProviderID>
2468
                 <CanonicalID>xri://@!1!2!3</CanonicalID>
                                                               ;XRDS #1 CID #3
2469
2470
                 <Service>
2471
                   <Type>http://openid.net/signon/1.0</Type>
2472
                   <Ref>xri://@x*y</Ref>
2473
                 </Service>
2474
              </XRD>
2475
              <XRDS ref="xri://@x*y">
2476
                 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2477
                   <Query>*x</Query>
2478
                   <ProviderID>xri://@</ProviderID>
2479
                   <CanonicalID>xri://@!7</CanonicalID>
                                                               ;XRDS #2 CID #1
2480
2481
                   <Service>
2482
                     <Type>xri://$res*auth*($v*2.0)</Type>
2483
                     <URI>http://x.example.com/</URI>
2484
                   </Service>
2485
                 </XRD>
2486
                 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2487
                   <Query>*y</Query>
2488
                   <ProviderID>xri://@!7</ProviderID>
2489
                   <CanonicalID>xri://@!7!8</CanonicalID>
                                                               ;XRDS #2 CID #2
2490
2491
                   <Service>
2492
                     <Type>xri://$res*auth*($v*2.0)</Type>
2493
                     <URI>http://y.example.com/</URI>
2494
                   </Service>
2495
                   <Service>
2496
                     <Type>http://openid.net/signon/1.0</Type>
2497
                     <URI>http://openid.example.com/</URI>
2498
                   </Service>
2499
                 </XRD>
2500
              </XRDS>
2501
           </XRDS>
```

# 12.6 Recursion and Backtracking

 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.

- 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.
- 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.
- 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.
- 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.
- 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.

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.

# **13 Service Endpoint Selection**

The second phase of XRI resolution is called *service endpoint selection*. As noted in Figure 2, this phase is invoked automatically for each iteration of authority resolution after the first in order to select the Next Authority Resolution Service Endpoint as defined in section 9.1.9. It is also performed after authority resolution is complete if optional service endpoint selection is requested.

# 13.1 Processing Rules

25392540

2541

2542

2543

2544

2545

25472548

2546 Figure 8 (non-normative) shows the overall logical flow of the service endpoint selection process.

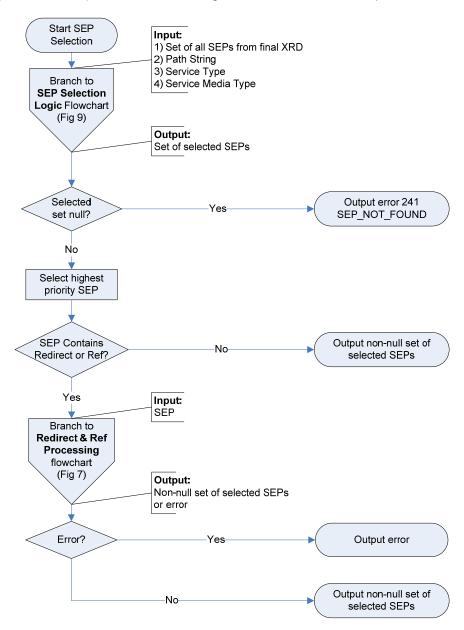


Figure 8: Service endpoint (SEP) selection flowchart.

- 2549 Following are the normative rules for the overall service endpoint selection process:
  - 1. The inputs for service endpoint selection are defined in Table 8.

2550

2551

2552 2553

25542555

25562557

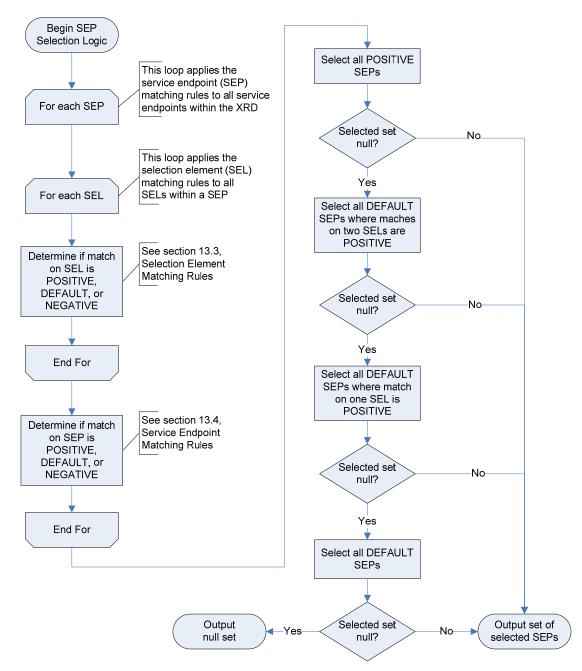
2558

2559 2560

- For the set of all service endpoints (xrd:XRD/xrd:Service elements) in the XRD, service endpoint selection MUST follow the logic defined in section 13.2. The output of this process MUST be either the null set or a selected set of one or more service endpoints.
- 3. If, after applying the service endpoint selection logic, the selected set is null, this function MUST return the error 241 SEP NOT FOUND.
- 4. If, after applying the service endpoint selection logic, the selected set is not null and the highest priority selected service endpoint contains an xrd:XRD/xrd:Service/xrd:Redirect or xrd:XRD/xrd:Service/xrd:Ref element, it MUST first be processed as specified in section 12. This is a recursive call that will produce a nested XRDS document as defined in section 12.5.

2569

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



2570 Figure 9: Service endpoint (SEP) selection logic flowchart.

2571 The following sections provide the normative rules for each section of this flowchart.

# 13.3 Selection Element Matching Rules

2573 The first set of rules govern the matching of selection elements.

2572

25742575

2576

2577 2578

25832584

2585

2586

2587 2588

2589

### 13.3.1 Selection Element Match Options

As defined in section 4.2.6, there are three categories of service endpoint selection elements: xrd:Type, xrd:Path, and xrd:MediaType. Within each service endpoint, there is a match option for each of the three categories of selection elements. Matches are tri-state: the three 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 match 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 match attribute is default 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

- 2579 Table 24: Match options for selection elements.
- 2580 The Precedence order is used in the Multiple Selection Element Matching Rule (section 13.3.5).
- IMPORTANT: Failure of a POSITIVE match does not necessarily mean a NEGATIVE match; it may still qualify as a DEFAULT match.

#### 13.3.2 The Match Attribute

All three service endpoint selection elements accept the optional match attribute. This attribute gives XRDS authors precise control over selection of SEPs based on the QXRI and other service endpoint selection parameters. An enumerated list of the values for the match attribute is defined in Table 25. If the match attribute is present with one of these values, the contents of the selection element MUST be ignored, and the corresponding matching rule MUST be applied. If 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 nodefault_t, nodefault_p or nodefault_m 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.	

2590 Table 25: Enumerated values of the global match attribute and corresponding matching rules.

- 2591 BACKWARDS COMPATIBILITY NOTE: earlier working drafts of this specification included the
- 2592 values match="none" and match="contents". Both are deprecated. The former is no longer
- 2593 supported and the latter is now the default behaviour of any selection element that does not
- 2594 include the match attribute. Implementers SHOULD accept these values accordingly.

### 13.3.3 Absent Selection Element Matching Rule

- 2596 If a service endpoint does not contain at least one instance of a particular category of selection
- 2597 element, it MUST be considered equivalent to the service endpoint having a DEFAULT match on
- 2598 that category of selection element UNLESS overriden by a nodefault\_\* parameter as specified
- 2599 in Table 25.

2595

2600

2610

2614

2615

2616

2617

2618

2619 2620

2621 2622

2623 2624

2625

2628

### 13.3.4 Empty Selection Element Matching Rule

- 2601 If a selection element is present in a service endpoint but the element is empty, and if the element
- 2602 does not contain a match attribute, it MUST be considered equivalent to having a match
- attribute with a value of null.

# 2604 13.3.5 Multiple Selection Element Matching Rule

- 2605 Each service endpoint has only one match option for each category of selection element.
- 2606 Therefore if a service endpoint contains more than one instance of the same category of selection
- 2607 element (i.e., more than one xrd: Type, xrd: Path, or xrd: Media Type element), the match for
- 2608 that category of selection element MUST be the match for the selection element(s) with the
- 2609 highest precedence match option as defined in Table 24.

### 13.3.6 Type Element Matching Rules

- The following rules apply to matching the value of the input Service Type parameter with the contents of a non-emtpy xrd:XRD/xrd:Service/xrd:Type element when its match attribute is absent.
  - 1. If the value is an XRI or IRI, it MUST be in URI-normal form as defined in section 4.4.
  - 2. Prior to comparsion (and only for the purpose of comparison), the values of the Service Type parameter and the xrd:XRD/xrd:Service/xrd:Type element SHOULD be normalized according to the requirements of their identifier scheme. In particular, if an XRI, IRI, or URI uses hierarchical syntax and does not include a local part (a path and/or query component) after the authority component, a trailing forward slash after the authority component MUST NOT be considered significant in comparisions. In all other cases, a trailing forward slash MUST be considered significant in comparisons unless this rule is overridden by scheme-specific comparision rules.
  - 3. To result in a POSITIVE match on this selection element, the values MUST be equivalent according to the equivalence rules of the applicable identifier scheme. Any other result is a NEGATIVE match on this selection element.
- As a best practice, service architects SHOULD assign identifiers for service types that are in URInormal form, do not require further normalization, and are easy to match.

### 13.3.7 Path Element Matching Rules

- The following rules apply to matching the value of the input Path String (the path portion of the QXRI as defined in section 8.1.1) with the contents of a non-empty
- 2632 xrd:XRD/xrd:Service/xrd:Path element when its match attribute is absent.
  - 1. If the value is a relative XRI or an IRI it MUST be in URI-normal form as defined in section 4.4.
  - 2. Prior to comparison, the leading forward slash separating an XRI authority component from the path component MUST be prepended to the Path String. Any subsequent forward slash, including trailing forward slashes, MUST be significant in comparisions.
  - 3. The contents of the xrd:XRD/xrd:Service/xrd:Path element SHOULD include the leading forward slash separating the XRI authority component from the path. If it does not, one MUST be prepended prior to comparision.
  - 4. Equivalence comparison SHOULD be performed using Caseless Matching as defined in section 3.13 of **[Unicode]**.
  - 5. To result in a POSITIVE match on this selection element, the value of the Path String MUST be a *subsegment stem match* with the contents of the xrd:XRD/xrd:Service/xrd:Path element. A subsegment stem match is defined as the entire Path String being character-for-character equivalent with any continuous sequence of subsegments or segments (including empty subsegments and empty segments) in the contents of the Path element beginning from the most significant (leftmost) subsegment. Subsegments and segments are formally defined in [XRISyntax]. Any other result MUST be a NEGATIVE match on this selection element.

XRI Resolution 2.0 Committee Specification 01 Copyright © OASIS® 1993–2008. All Rights Reserved.

### Examples of this rule are shown in Table 26.

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

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

2654

# 13.3.8 MediaType Element Matching Rules

2655

2659

2660

2661 2662

2663

2664

2665

2666

2667

2668

2671

2674

2681

2686

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

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

# 13.4 Service Endpoint Matching Rules

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

# 13.4.1 Service Endpoint Match Options

2672 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.	

2673 Table 27: Match options for service endpoints.

#### 13.4.2 Select Attribute Match Rule

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

#### 13.4.3 All Positive Match Rule

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

#### 13.4.4 Default Match Rule

2687 If a service endpoint fails the Select Attribute Match Rule and the All Positive Match Rule, but
2688 none of the three categories of selection elements has a NEGATIVE match as defined in section
2689 13.3, the service endpoint MUST be a DEFAULT match.

# 2690 13.5 Service Endpoint Selection Rules

2691 The final set of rules governs the selection of service endpoints based on their matches.

#### 2692 13.5.1 Positive Match Rule

- After applying the matching rules to service endpoints in section 13.4, all service endpoints that
- 2694 have a POSITIVE match MUST be selected. Only if there are no service endpoints with a
- 2695 POSITIVE match is the Default Match Rule invoked.

#### 2696 13.5.2 Default Match Rule

- If the Positive Match Rule above fails, then the service endpoints with a DEFAULT match that have the highest number of POSITIVE matches on each category of selection element MUST be selected. This means:
  - The service endpoints in the DEFAULT set that have two POSITIVE selection element matches MUST be selected.
  - 2. If the previous set is empty, the service endpoints in the DEFAULT set that have one POSITIVE selection element match MUST be selected.
- 3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.
- 2705 4. If the previous set is empty, no service endpoint is selected and the return set is null.

### 2706 13.6 Pseudocode

2700

2701

2702

2703

- 2707 The following pseudocode provides a precise description of the service endpoint selection logic.
- 2708 The pseudocode is normative, however if there is a conflict between it and the rules stated in the
- 2709 preceeding sections, the preceeding sections shall prevail.
- 2710 The pseudocode uses nine Boolean flags to record the match state for each category of selection
- 2711 element (SEL) in a service endpoint (SEP):
- Postive.x (where x = Type, Path, or MediaType)
- Default.x (where x = Type, Path, or MediaType)
- Present.x (where x = Type, Path, or MediaType)
- 2715 The variable Nodefault.x refers to the value of the nodefault\_t (Type), nodefault\_p
- 2716 (Path), and nodefault\_m (MediaType) subparameters as explained in Table 25.
- Note that the complete set of nine SEL match flags is needed for each SEP. The pseudocode first
- 2718 does a loop through all SEPs in the XRD to:
- 2719 1. Set the SEL match flags according to the rules specified in section 13.3;
- 2720 2. Process the SEL match flags to apply the SEP matching rules specified in section 13.4;
- 3. Apply the positive SEP selection rule specified in section 13.5.1.
- 2722 After this loop is complete, the pseudocode tests to see if default SEP selection processing is
- 2723 required. If so, it performs a second loop applying the default SEP selection rules specified in
- 2724 section 13.5.2.

- 2725 NOTE: In this pseudocode, when the words POSITIVE, DEFAULT, or NEGATIVE appear in
- 2726 UPPERCASE, they refer to the SEL match type or SEP match type as defined in Table 24 and
- Table 27. When they appear in First Letter Caps, they refer to the Boolean flags defined above.

```
2729
```

```
2730
            FOR EACH SEP
2731
               CREATE set of nine SEL match flags (see text above)
2732
               SET all flags to FALSE
2733
               FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
2734
                      SET Present.x=TRUE
2735
                      IF match type on this SEL is POSITIVE
2736
                             IF select="true"
                                                                  ;see 13.4.2
2737
                                     ADD SEP TO SELECTED SET
2738
                                    NEXT SEP
2739
                             ELSE
2740
                                     SET Positive.x=TRUE
2741
                             ENDIF
2742
                      ELSEIF match="default"
                                                                  ;see 13.3.2
2743
                             IF Positive.x != TRUE AND
                                                                  ;see 13.3.5
2744
                             Nodefault.x != TRUE
                                                                  ;see 13.3.2
2745
                                    SET Default.x=TRUE
2746
                             ENDIF
2747
                      ENDIF
2748
               ENDFOR
2749
               FOR EACH category x (where x=Type, Path, or Mediatype)
2750
                      IF Present.x=FALSE
                                                                  ;see 13.3.3
2751
                             IF Nodefault.x != TRUE
                                                                  ;see 13.3.2
2752
                                    SET Default.x=TRUE
2753
                             ENDIF
2754
                      ENDIF
2755
               ENDFOR
2756
               IF Positive.Type=TRUE AND
2757
                  Positive.Path=TRUE AND
2758
                                                                  ;see 13.4.3
                  Positive.Mediatype=TRUE
2759
                      ADD SEP TO SELECTED SET
2760
                      MEYT SED
2761
               ELSEIF SELECTED SET != EMPTY
                                                                  ;see 13.5.1
2762
                     NEXT SEP
2763
               ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
2764
                      (Positive.Path=TRUE OR Default.Path=TRUE) AND
2765
                       (Positive.MediaType=TRUE OR Default.MediaType=TRUE)
2766
                      ADD SEP TO DEFAULT SET
                                                                  ;see 13.4.4
2767
               ENDIF
2768
            ENDFOR
2769
            IF SELECTED SET = EMPTY
2770
               FOR EACH SEP IN DEFAULT SET
                                                                  ;see 13.5.2
2771
                      IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
2772
                       (Positive.Type=TRUE AND Positive.MediaType=TRUE) OR
2773
                       (Positive.Path=TRUE AND Positive.MediaType=TRUE)
2774
                             ADD SEP TO SELECTED SET
2775
                      ENDIF
2776
               ENDFOR
2777
               IF SELECTED SET = EMPTY
2778
                      FOR EACH SEP IN DEFAULT SET
                                                                 ;see 13.5.2
2779
                             IF Positive.Type=TRUE OR
2780
                             Positive.Path=TRUE OR
2781
2782
                             Positive.MediaType=TRUE
                                     ADD SEP TO SELECTED SET
2783
                             ENDIF
2784
                      ENDFOR
2785
               ENDIF
2786
            ENDIF
2787
            IF SELECTED SET != EMPTY
2788
               RETURN SELECTED SET
2789
            ELSE
2790
              RETURN DEFAULT SET
2791
            ENDIF
```

# 2792 13.7 Construction of Service Endpoint URIs

- The final step in the service endpoint selection process is construction of the service endpoint URI(s). This step is necessary if either:
- The resolution output format is a URI List.
- Automatic URI construction is requested using the uric parameter.

### 13.7.1 The append Attribute

2797

2798

2799

2805

2806 2807

2808

2809

2810

2811

2812 2813

2814

The append attribute of a xrd:XRD/xrd:Service/xrd:URI element is used to specify how 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 append 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 including the leading forward slash
	b) If only a query is present, the Query String including the leading question mark
	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) not including the trailing forward slash
path	Path String including the leading forward slash
query	Query String including the leading question mark
qxri	Entire QXRI

- 2800 Table 28: Values of the append attribute and the corresponding QXRI component to append.
- 2801 If the append attribute is absent, the default value is none. Following are the rules for construction of the final service endpoint URI based on the value of the append attribute.
- 2803 IMPORTANT: Implementers must follow these rules exactly in order to give XRDS authors precise control over construction of service endpoint URIs.
  - 1. If the value is none, the exact contents of the xrd:URI element MUST be returned directly without any further processing.
  - 2. For any other value, the exact value in URI-normal form of the QXRI component specified in Table 28, including any leading delimiter(s) and without any additional escaping or percent encoding MUST be appended directly to the exact contents of the xrd:URI element including any trailing delimiter(s). If the value of the QXRI component specified in Table 28 consists of only a leading delimiter, then this value MUST be appended according to these rules. If the value of the QXRI component specified in Table 28 is null, then the contents of the xrd:URI element MUST be returned directly exactly as if the value of the append attribute was none.

- 3. If any HXRI query parameters for proxy resolution were added to an existing QXRI query component as defined in section 11.3, these query parameters MUST be removed prior to performing the append operation as also defined in section 11.3. In particular, if after removal of these query parameters the QXRI query component consists of only a string of one or more question marks (the delimiting question mark plus zero or more additional question marks) then exactly one question mark MUST also be removed. This preserves the query component of the original QXRI if it was null or contained only question marks.
- IMPORTANT: Construction of HTTP(S) URIs for authority resolution service endpoints is defined in section 9.1.10. Note that this involves an additional step taken after all URI construction steps specified in this section are complete. In other words, if the URI element of an authority resolution service endpoint includes an append attribute, the Next Authority Resolution Service URI MUST be fully constructed according to the algorithm in this section before appending the Next Authority String as defined in section 9.1.10.
- WARNING: Use of any value of the append attribute other than authority on the URI element for an authority resolution service endpoint is NOT RECOMMENDED due to the complexity it introduces.

#### 13.7.2 The uric Parameter

2831

2835

2836

2837

2838

2839

2840 2841

2842 2843

- The uric subparameter of the Resolution Output Format is used to govern whether a resolver should perform construction of the URI automatically on behalf of a consuming application. Following are the processing rules for this parameter:
  - 1. If uric=true, a resolver MUST apply the URI construction rules specified in section 13.7.1 to each xrd:XRD/xrd:Service/xrd:URI element in the final XRD in the resolution chain. Note that this step is identical to the processing a resolver must perform to output a URI list.
  - 2. The resolver MUST replace the value of each xrd:XRD/xrd:Service/xrd:URI element in the final XRD with the fully constructed URI value.
  - 3. The resolver MUST subsequently remove the append attribute from each xrd:XRD/xrd:Service/xrd:URI element in the final XRD.
  - 4. If uric=false or the parameter is absent or empty, a resolver MUST NOT perform any of the processing specified in this section.

# 14 Synonym Verification

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

### 14.1 Redirect Verification

2845

2849 2850

2851

2852

2853

2854

2855

2856

2857

2858

2859

2860

2862

2865

2866 2867

2868 2869

2870

2871

2872

2873

2875

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

- If resolution of the Redirect succeeds, the resolver MUST first verify that the set of XRD synonym elements (as specified in section 5.2) contained in the new XRD are equivalent to or a subset of those contained in the XRD containing the Redirect.
- Secondly, the resolver MUST verify that the content of each synonym element contained in the new XRD is exactly equivalent to the content of the corresponding element in the XRD containing the Redirect.
- 3. If either rule above fails, the resolver MUST stop and return the error 253 REDIRECT\_VERIFY\_FAILED in the XRD where the error occurred or as a plain text error message as defined in section 15.
- 2861 For examples see section 12.5.1.

# 14.2 EquivID Verification

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

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

#### 2874 **Example**:

- Fully-Qualified Query Identifier: http://example.com/user
- **2876** Asserted EquivID: xri://=!1000.c78d.402a.8824.bf20

2877 First XRDS (for http://example.com/user — simplified for illustration purposes):

```
2878
            <XRDS>
2879
2880
                <EquivID>xri://=!1000.c78d.402a.8824.bf20</EquivID>
2881
                <CanonicalID>http://example.com/user</CanonicalID>
2882
                <Service priority="10">
2883
                   . . .
2884
                </Service>
2885
2886
              </XRD>
2887
            </XRDS>
```

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

```
2889
            <XRDS>
2890
              <XRD>
2891
                <Query>!1000.c78d.402a.8824.bf20</Query>
2892
                <ProviderID>xri://=</ProviderID>
2893
                <EquivID>http://example.com/user</EquivID>
2894
                <CanonicalID>xri://=!1000.c78d.402a.8824.bf20</CanonicalID>
2895
                <Service priority="10">
2896
2897
                </Service>
2898
                . . .
2899
              </XRD>
2900
           </XRDS>
```

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

### 14.3 CanonicalID Verification

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

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

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

# 2933 14.3.1 HTTP(S) URI Verification Rules

- To verify that an HTTP(S) URI is a valid CanonicalID synonym for a fully-qualified query identifier (defined in section 5.1), a resolver MUST verify that all the following tests are successful:
  - 1. The fully-qualified query identifier MUST also be an HTTP(S) URI.
  - 2. The query identifier MUST be resolved as specified in section 6.
  - 3. The asserted CanonicalID synonym MUST be an HTTP(S) URI equivalent to: a) the fully-qualified query identifier, or b) the fully-qualified query identifier plus a valid fragment as defined by [RFC3986].
- See the example in section 14.3.5.

29362937

2938

2939 2940

29422943

2944 2945

2946

2947

2948

2949

2950

2951

2952

2953

2954

2955

2956

2957

2958

2959

2960 2961

2962

29632964

2965

2966

2967

2968

2969

#### 14.3.2 XRI Verification Rules

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

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

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

# 14.3.3 Canonical EquivID Verification

- CanonicalID verification also requires verification of a CanonicalEquivID *only if it is present in the*final XRD in the resolution chain. Since CanonicalEquivID verification typically requires an extra
- resolution cycle, restricting automatic verification to the final XRD in the resolution chain ensures
- it will add at most one additional resolution cycle.
- 2974 CanonicalEquivID verification MUST NOT be performed unless CanonicalID verification as
- 2975 specified in section 14.3 has completed successfully. The resulting value is called the *verified*
- 2976 CanonicalID.

- To verify that a CanonicalEquivID is an authorized synonym for a verified CanonicalEquivID, a resolver MUST verify that either: a) the value of the CanonicalEquivID element is character-by-character equivalent to the verified CanonicalID (since both appear in the same XRD, all other normalization rules are waived), or b) that all the following tests are successful:
  - 1. The asserted Canonical EquivID value MUST be a valid HTTP(S) URI or XRI.
  - 2. The asserted CanonicalEquivID value MUST resolve successfully to an XRDS document according to the rules in this specification using the same resolution parameters as in the original resolution request.
  - 3. The CanonicalID in the final XRD of the resolved XRDS document MUST be verified and MUST be equivalent to the asserted CanonicalEquivID.
  - 4. The final XRD in the resolved XRDS document MUST contain either an EquivID or a CanonicalEquivID "backpointer" whose value is equivalent to the verified CanonicalID in the XRD asserting the CanonicalEquivID.
- SPECIAL SECURITY CONSIDERATION: See section 5.2.2 regarding the rules for provisioning of xrd:XRD/xrd:EquivID and xrd:XRD/xrd:CanonicalEquivID elements in an XRD.

#### 14.3.4 Verification Status Attributes

- If CanonicalID verification is performed, an XRI resolver MUST return the CanonicalID and CanonicalEquivID verification status using an attribute of the xrd:XRD/xrd:Status element in each XRD in the output as follows:
  - 1. CanonicalID verification MUST be reported using the cid attribute.
- 2997 2. Canonical EquivID verification MUST be reported using the ceid attribute.
  - 3. Both attributes accept four enumerated values: absent if the element is not present, off if verification is not performed, verified if the element is verified, and failed if verification fails.
  - 4. The off value applies to both elements if CanonicalID verification is not performed (cid=false).
  - 5. The off value applies to the CanonicalEquivID element in any XRD before the final XRD if CanonicalID verification is performed (cid=true), because a resolver only verifies this element in the final XRD.
  - 6. If cid=true and verification of any CanonicalID element fails, *verification of all CanonicalIDs in all subsequent XRDs in the same XRDS document MUST fail.*
- From these verification status attributes, a consuming application can confirm on every XRD in the XRDS document whether the CanonicalID is present and has been verified. In addition, for the final XRD in the XRDS document, it can confirm whether the CanonicalEquivID element is present and has been verified.

2981

2982

2983 2984

2985

2986

2987 2988

2989

2992

2996

2998

2999

3000

3001

3002

3003

3004

3005 3006

### 14.3.5 Examples

### 3014 **Example #1:**

3013

- Fully-Qualified Query Identifier: http://example.com/user
- Asserted CanonicalID: http://example.com/user#1234
- 3017 XRDS (simplified for illustration purposes):

```
3018
            <XRDS ref="http://example.com/user">
3019
              <XRD>
3020
                <CanonicalID>http://example.com/user#1234</CanonicalID>
3021
                <Service priority="10">
3022
3023
                </Service>
3024
                . . .
3025
              </XRD>
3026
            </XRDS>
```

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

3028

3029

3030

3031

3032

3027

#### Example #2:

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

XRDS (for =example.name\*delegate.name):

```
3033
            <XRDS ref="xri://=example.name*delegate.name">
3034
              <XRD>
3035
                <Ouerv>*example.name</Ouerv>
3036
                <ProviderID>xri://=</ProviderID>
3037
                <LocalID>!1000.62b1.44fd.2855</LocalID>
3038
                <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3039
                <Service>
3040
                   <ProviderID>xri://=!1000.62b1.44fd.2855</providerID>
3041
                   <Type>xri://$res*auth*($v*2.0)</Type>
3042
                   <MediaType>application/xrds+xml</MediaType>
3043
                   <URI priority="10">http://resolve.example.com</URI>
3044
                   <URI priority="15">http://resolve2.example.com</URI>
3045
                   <URI>https://resolve.example.com</URI>
3046
                </Service>
3047
3048
             </XRD>
3049
             <XRD>
3050
               <Query>*delegate.name</Query>
3051
               <ProviderID>xri://=!1000.62b1.44fd.2855</providerID>
3052
               <LocalID>!1234</LocalID>
3053
               <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
3054
                <Service priority="1">
3055
3056
                </Service>
3057
                . . .
3058
              </XRD>
3059
           </XRDS>
```

The asserted CanonicalID satisifies the XRI verification rules in section 14.3.2.

```
3062 Example #3:
```

- Fully-Qualified Query Identifier: http://example.com/user
- Asserted CanonicalID: http://example.com/user
- Asserted Canonical EquivID: https://different.example.net/path/user

3066 First XRDS (for http://example.com/user):

```
3067
            <XRDS ref="http://example.com/user">
3068
              <XRD>
3069
                <CanonicalID>http://example.com/user</CanonicalID>
3070
                <CanonicalEquivID>
3071
                https://different.example.net/path/user
3072
              </CanonicalEquivID>
3073
                <Service priority="10">
3074
3075
                </Service>
3076
3077
              </XRD>
3078
            </XRDS>
```

3079 Second XRDS (for https://different.example.net/path/user):

```
3080
           <XRDS ref="https://different.example.net/path/user">
3081
              <XRD>
3082
                <EquivID>http://example.com/user</EquivID>
3083
                <CanonicalID>https://different.example.net/path/user</CanonicalID>
3084
                <Service priority="10">
3085
3086
                </Service>
3087
3088
              </XRD>
3089
           </XRDS>
```

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

3094 **Example #4:** 

3090

3091 3092

3093

3095

- Fully-Qualified Query Identifier: http://example.com/user
- **3096** Asserted CanonicallD: http://example.com/user
- 3097 Asserted Canonical EquivID: = ! 1000.62b1.44fd.2855

3098 XRDS (for http://example.com/user):

```
3099
            <XRDS ref="http://example.com/user">
3100
              <XRD>
3101
                <CanonicalID>http://example.com/user</CanonicalID>
3102
                <CanonicalEquivID>xri://=!1000.62b1.44fd.2855</CanonicalEquivID>
3103
                <Service priority="10">
3104
                   . . .
3105
                </Service>
3106
3107
              </XRD>
3108
            </XRDS>
```

```
3110
       XRDS (for xri://=!1000.62b1.44fd.2855):
3111
           <XRDS ref="xri://=!1000.62b1.44fd.2855">
3112
3113
               <Query>!1000.62b1.44fd.2855
3114
               <ProviderID>xri://=</ProviderID>
3115
               <EquivID>http://example.com/user</EquivID>
3116
               <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3117
               <Service priority="10">
3118
3119
               </Service>
3120
               . . .
3121
             </XRD>
3122
           </XRDS>
```

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

3125 3126

3127

3128

3123

3124

#### Example #5:

- Fully-Qualified Query Identifier: =example.name
- 3129 Asserted CanonicalID: xri://=!1000.62b1.44fd.2855
- 3130 Asserted Canonical EquivID: https://example.com/user

3131 First XRDS (for =example.name):

```
3132
           <XRDS ref="xri://=example.name">
3133
              <XRD>
3134
                <Query>*example.name</Query>
3135
                <ProviderID>xri://=</ProviderID>
3136
                <LocalID>!1000.62b1.44fd.2855</LocalID>
3137
                <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3138
                <CanonicalEquivID>https://example.com/user</CanonicalEquivID>
3139
                <Service priority="10">
3140
3141
                </Service>
3142
3143
              </XRD>
3144
           </XRDS>
```

3145 Second XRDS (for https://example.com/user):

```
3146
            <XRDS ref="https://example.com/user">
3147
              <XRD>
3148
                <EquivID>xri://=!1000.62b1.44fd.2855</EquivID>
3149
                <CanonicalID>https://example.com/user</CanonicalID>
3150
                <Service priority="10">
3151
                   . . .
3152
                </Service>
3153
3154
              </XRD>
3155
            </XRDS>
```

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

3158 3159

3156 3157

#### 3161 **Example #6:**

3165

3196

- Fully-Qualified Query Identifier: =example.name\*delegate.name
- 3163 Asserted CanonicalID: xri://=!1000.62b1.44fd.2855!1234
- Asserted Canonical Equiv ID: @!1000.f3da.9056.aca3!5555

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

```
3166
            <XRDS ref="xri://=example.name*delegate.name">
3167
              <XRD>
3168
                <Query>*example.name</Query>
3169
                <ProviderID>xri://=</ProviderID>
3170
                <LocalID>!1000.62b1.44fd.2855</LocalID>
3171
                <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3172
                <Service>
3173
                   <ProviderID>xri://=!1000.62b1.44fd.2855</providerID>
3174
                   <Type>xri://$res*auth*($v*2.0)</Type>
3175
                   <MediaType>application/xrds+xml</MediaType>
3176
                   <URI priority="10">http://resolve.example.com</URI>
3177
                   <URI priority="15">http://resolve2.example.com</URI>
3178
                   <URI>https://resolve.example.com</URI>
3179
                </Service>
3180
3181
             </XRD>
3182
             <XRD>
3183
               <Query>*delegate.name</Query>
3184
               <ProviderID>xri://=!1000.62b1.44fd.2855</providerID>
3185
               <LocalID>!1234</LocalID>
3186
               <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
3187
               <CanonicalEquivID>
3188
                xri://@11000.f3da.9056.aca3!5555
3189
                </CanonicalEquivID>
3190
                <Service priority="1">
3191
                   . . .
3192
                </Service>
3193
3194
              </XRD>
3195
           </XRDS>
```

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

```
3197
           <XRDS ref="xri://@!1000.f3da.9056.aca3!5555">
3198
3199
               <Query>!1000.f3da.9056.aca3/Ouery>
3200
               <ProviderID>xri://@</ProviderID>
3201
               <CanonicalID>xri://@!1000.f3da.9056.aca3</CanonicalID>
3202
3203
                  <ProviderID>xri://@!1000.f3da.9056.aca3</providerID>
3204
                  <Type>xri://$res*auth*($v*2.0)</Type>
3205
                  <MediaType>application/xrds+xml</MediaType>
3206
                  <URI priority="10">http://resolve.example.com</URI>
3207
                  <URI priority="15">http://resolve2.example.com</URI>
3208
                  <URI>https://resolve.example.com</URI>
3209
               </Service>
3210
3211
             </XRD>
3212
             <XBD>
3213
               <Query>!5555</Query>
3214
               <ProviderID>xri://@!1000.f3da.9056.aca3</providerID>
3215
               <LocalID>!5555</LocalID>
3216
               <EquivID>xri://=!1000.62b1.44fd.2855!1234</EquivID>
```

The CanonicalEquivID asserted in the final XRD of the first XRDS satisifies the verification rules in section 14.3.3 because it resolves to a second XRDS whose final XRD asserts an EquivID backpointer to the CanonicalID of the final XRD in the first XRDS.

# 15 Status Codes and Error Processing

#### 3228 15.1 Status Elements

3227

3232

3233

3234 3235

3237

3238

3239

3244

3245

3246

3247

3248

3249 3250

3251

3252

3253

3254

3255

3256

3257

- 3229 XRDS architecture uses two XRD elements for status reporting:
- 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.
  - 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).
- 3236 Following are the normative rules that apply to usage of these elements:
  - 1. For XRDS servers and clients, each of these elements is OPTIONAL.
  - 2. An XRI authority server is REQUIRED to include an xrd:XRD/xrd:ServerStatus element for each XRD in a resolution response.
- 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.
  - 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.
    - 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.
    - 5. These elements MUST include the status codes specified in section 15.2 as the value of the required code attribute.
  - 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.

#### 15.2 Status Codes

- 3258 XRI resolution status codes are patterned after the HTTP model. They are broken into three 3259 major categories:
- 1xx: Success—the requested resolution operation was completed successfully.
- 2xx: Permanent errors—the resolver encountered an error from which it could not recover.
- 3xx: Temporary errors—the resolver encountered an error condition that may be only temporary.

3265 The 2xx and 3xx categoryes are broken into seven minor categories:

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

3267 • x1x: Input error

3268 • x2x: Generic authority resolution error.

• x3x: Trusted authority resolution error.

3270 • x4x: Service endpoint (SEP) selection error.

3271 • x5x: Redirect error.

3272 • x6x: Ref error.

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

Code	Symbolic Status	Phase(s)	Description
100	SUCCESS	Any	Operation was successful.
200	PERM_FAIL	Any	Generic permanent failure.
201	NOT_IMPLEMENTED	Any	The requested function (trusted resolution, service endpoint selection) is not implement by the resolver.
202	LIMIT_EXCEEDED	Any	A locally configured resource limit was exceeded. Examples: number of 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 xrd: Query 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	https=true but a Redirect element containing an HTTPS URI was not found.
253	REDIRECT_VERIFY_FAILED	Redirect Processing	Synonym verification failed in an XRD after following a redirect. See section 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 refs parameter was set to false.
300	TEMPORARY_FAIL	Any	Generic temporary failure.
301	TIMEOUT_ERROR	Any	Locally-defined timeout limit has lapsed during an operation (e.g. network latency).
320	NETWORK_ERROR	Authority resolution	Generic error during authority resolution phase (includes uncaught exception, system error, network error).
321	UNEXPECTED_RESPONSE	Authority resolution	When querying an authority 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).

Table 29: Error codes for XRI resolution.

## 15.3 Status Context Strings

3278

3286

3297

- 3279 Each status code in Table 29 MAY be returned with an optional status context string that provides
- 3280 additional human-readable information about the status or error condition. When the Resolution
- 3281 Output Format is an XRDS document or XRD element, this string is returned as the contents of
- 3282 the xrd:XRD/xrd:ServerStatus and xrd:XRD/xrd:Status elements. When the
- 3283 Resolution Output Format is a URI List, this string MUST be returned as specified in section 15.4.
- 3284 Implementers SHOULD provide error context strings with additional information about an error
- 3285 and possible solutions whenever it can be helpful to developers or end users.

## 15.4 Returning Errors in Plain Text or HTML

- If the Resolution Output Format is a URI List as defined in section 8.2, an error MUST be returned with the content type text/plain. In this content:
- The first line MUST consist of only the numeric error code as defined in section 15.2 followed by a CRLF.
- The second line is RECOMMENDED; if present it MUST contain the error context string as defined in section 15.3.
- 3293 The same rules apply if the Resolution Output Format is an HTTP(S) Redirect as defined in
- 3294 section 8.2, except the media type MAY also be text/html. It is particularly important in this
- case to return an error message that will be understandable to an end-user who may have no
- 3296 knowledge of XRI resolution or the fact that the error is coming from an XRI proxy resolver.

## 15.5 Error Handling in Recursing and Proxy Resolution

- 3298 In recursing and proxy resolution (sections 9.1.8 and 11), a server is acting as a client resolver for
- 3299 other authority resolution service endpoints. If in this intermediary capacity it receives an
- 3300 unrecoverable error, it MUST return the error to the originating client in the output format
- 3301 specified by the value of the requested Resolution Output Format as defined in section 8.2.
- 3302 If the output format is an XRDS document, it MUST contain xrd: XRD elements for all
- 3303 subsegments successfully resolved or retrieved from cache prior to the error. Each XRD MUST
- 3304 include the xrd: ServerStatus element as reported by the authoritative server. The final
- 3305 xrd: XRD element MUST include the xrd: Query element that produced the error and the
- 3306 xrd: Status element that describes the error as defined above.
- 3307 If the output format is an XRD element, it MUST include the xrd:Query element that produced
- 3308 the error, the xrd: ServerStatus element as reported by the authoritative server, and the
- 3309 xrd:Status element that describes the error as defined above.
- 3310 If this output format is a URI List or an HTTP(S) redirect, a proxy resolver SHOULD return a
- human-readable error message as specified in section 15.4.

#### 16Use of HTTP(S) 3312 16.1 HTTP Errors 3313 3314 When a resolver encounters fatal HTTP(S) errors during the resolution process, it MUST return 3315 the appropriate XRI resolution error code and error message as defined in section 15. In this way calling applications do not have to deal separately with XRI and HTTP error messages. 3316 16.2 HTTP Headers 3317 3318 **16.2.1 Caching** 3319 The HTTP caching capabilities described by [RFC2616] should be leveraged for all XRDS and 3320 XRI resolution protocols. Specifically, implementations SHOULD implement the caching model 3321 described in section 13 of [RFC2616], and in particular, the "Expiration Model" of section 13.2, as 3322 this requires the fewest round-trip network connections. 3323 All XRI resolution servers SHOULD send the Cache-Control or Expires headers in their 3324 responses per section 13.2 of [RFC2616] unless there are overriding security or policy reasons to 3325 omit them. 3326 Note that HTTP Cache headers SHOULD NOT conflict with expiration information in an XRD. 3327 That is, the expiration date specified by HTTP caching headers SHOULD NOT be later than any 3328 of the expiration dates for any of the xrd: Expires elements returned in the HTTP response. 3329 This implies that recursing and proxy resolvers SHOULD compute the "soonest" expiration date 3330 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching 3331 headers for the HTTP response. 16.2.2 Location 3332 3333 During HTTP interaction, "Location" headers may be present per [RFC2616] (i.e., during 3XX 3334 redirects). Redirects SHOULD be made cacheable through appropriate HTTP headers, as specified in section 16.2.1. 3335 16.2.3 Content-Type 3336 3337 For authority resolution, the Content-Type header in the 2XX responses MUST contain the media 3338 type identifier values specified in Table 11 (for generic resolution), Table 15 (for HTTPS trusted 3339 resolution), Table 16 (for SAML trusted resolution), or Table 17 (or HTTPS+SAML trusted 3340 resolution). 3341 Following the optional service endpoint selection phase, clients and servers MAY negotiate 3342 content type using standard HTTP content negotiation features. Regardless of whether this 3343 feature is used, however, the server MUST respond with an appropriate media type in the 3344 Content-Type header if the resource is found and an appropriate content type is returned. **16.3 Other HTTP Features** 3345 3346 HTTP provides a number of other features including transfer-coding, proxying, validation-model 3347 caching, and so forth. All these features may be used insofar as they do not conflict with the

required uses of HTTP described in this document.

## 16.4 Caching and Efficiency

## 16.4.1 Resolver Caching

3349

3350

3373 3374

3375

3376

3377

3378 3379

3380

3381

3382

3383

3384

3385

3386

3387 3388

3389

3390

- In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the application level. For best results, however, resolution clients SHOULD be conservative with caching expiration semantics, including cache expiration dates. This implies that in a series of HTTP redirects, for example, the results of the entire process SHOULD only be cached as long as the shortest period of time allowed by any of the intermediate HTTP responses.
- 3356 Because not all HTTP client libraries expose caching expiration to applications, identifier
  3357 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the
  3358 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments
  3359 should be mindful of limitations in current HTTP clients and proxies.
- The cache expiration time of an XRD may also be explicitly limited by the parent authority. If the expiration time in the xrd:Expires element is sooner than the expiration time calculated from the HTTP caching semantics, the XRD MUST be discarded before the expiration time in xrd:Expires. Note also that a saml:Assertion element returned during SAML trusted resolution has its own signature expiration semantics as defined in [SAML]. While this may invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if it is not expired by HTTP caching semantics or the xrd:Expires element.
- With both application-level and HTTP-level caching, the resolution process is designed to have minimal overhead. Resolution of each qualified subsegment of an XRI authority component is a separate step described by a separate XRD, so intermediate results can typically be cached in their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified subsegments, which are common to more identifiers, will naturally result in a greater number of cache hits than resolution of lower-level subsegments.

## 16.4.2 Synonyms

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

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

# 17 Extensibility and Versioning

## 17.1 Extensibility

#### 17.1.1 Extensibility of XRDs

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

Extension elements MUST NOT require new interpretation of elements defined in this document.

If an extension element is present, a processor MUST be able to ignore it and still correctly process the XRDS document.

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

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

```
3414
            <XRD>
3415
              <Service>
3416
3417
              </Service>
3418
              <other:SuperService>
3419
                <Service>
3420
3421
                  <other:ExtensionElement>...</other:ExtensionElement>
3422
                </Service>
3423
              </other:SuperService>
3424
```

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

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

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

3436 xrd:XRD/xrd:Type element. However XRD processors MAY ignore instances of this element and process the XRD as specified in this document.

## 17.1.2 Other Points of Extensibility

- The use of HTTP(S), XML, XRIs, and URIs in the design of XRDS documents, XRD elements, and XRI resolution architecture provides additional specific points of extensibility:
- Specification of new resolution service types or other service types using XRIs, IRIs, or URIs as values of the xrd:Type element.
- Specification of new resolution output formats or features using media types and media type parameters as values of the xrd:MediaType element as defined in [RFC2045] and [RFC2046].
- HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- Use of cross-references within XRIs, particularly for associating new types of metadata with a resource. See [XRISyntax] and [XRIMetadata].

## 17.2 Versioning

3438

3450

3455

3456

3457

3458 3459

3460

3461

3462

3468

3473

- Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary, this section describes versioning guidelines.
- In general, this specification follows the same versioning guidelines as established in section 4.2.1 of **[SAML]**:
  - In general, maintaining namespace stability while adding or changing the content of a schema are competing goals. While certain design strategies can facilitate such changes, it is complex to predict how older implementations will react to any given change, making forward compatibility difficult to achieve. Nevertheless, the right to make such changes in minor revisions is reserved, in the interest of namespace stability. Except in special circumstances (for example, to correct major deficiencies or to fix errors), implementations should expect forward-compatible schema changes in minor revisions, allowing new messages to validate against older schemas.
- Implementations SHOULD expect and be prepared to deal with new extensions and message types in accordance with the processing rules laid out for those types. Minor revisions MAY introduce new types that leverage the extension facilities described in [this section]. Older implementations SHOULD reject such extensions gracefully when they are encountered in contexts that dictate mandatory semantics.

## 17.2.1 Version Numbering

- Specifications from the OASIS XRI Technical Committee use a Major and Minor version number expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version number *MajorA.MinorA* if and only if:
- 3472 Major<sub>B</sub> > Major<sub>A</sub> OR ( ( Major<sub>B</sub> = Major<sub>A</sub> ) AND Minor<sub>B</sub> > Minor<sub>A</sub> )

## 17.2.2 Versioning of the XRI Resolution Specification

- New releases of the XRI Resolution specification may specify changes to the resolution protocols
- 3475 and/or the XRD schema in Appendix B. When changes affect either of these, the resolution
- 3476 service type version number will be changed. Where changes are purely editorial, the version
- 3477 number will not be changed.

3478 3479 3480	In general, if a change is backward-compatible, the new version will be identified using the current major version number and a new minor version number. If the change is not backward-compatible, the new version will be identified with a new major version number.
3481	17.2.3 Versioning of Protocols
3482 3483 3484 3485	The protocols defined in this document may also be versioned by future releases of the XRI Resolution specification. If these protocols are not backward-compatible with older implementations, they will be assigned a new XRI with a new version identifier for use in identifying their service type in XRDs. See section 3.1.2.
3486 3487 3488 3489 3490	Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely to continue to use the same XRI to identify the protocol as was used in previous versions of the XRI Resolution specification.
3491	17.2.4 Versioning of XRDs
3492 3493 3494	The $xrd: XRDS$ document element is intended to be a completely generic container, i.e., to have no specific knowledge of the elements it may contain. Therefore it has no version indicator, and can remain stable indefinitely because there is no need to version its namespace.
3495 3496 3497 3498	The xrd:XRD element has a version attribute. This attribute is OPTIONAL for this version of the XRI resolution specification (version 2.0). This attribute will be REQUIRED for all future versions of this specification. When used, the value of this attribute MUST be the exact numeric version value of the XRI Resolution specification to which its containing elements conform.
3499 3500 3501 3502	When new versions of the XRI Resolution specification are released, the namespace for the XRD schema may or may not be changed. If there is a major version number change, the namespace for the $xrd:XRD$ schema is likely to change. If there is only a minor version number change, the namespace for the $xrd:XRD$ schema may remain unchanged.

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

3505	18Security and Data Protection
3506 3507 3508 3509	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.
3510	18.1 DNS Spoofing or Poisoning
3511 3512 3513 3514 3515 3516 3517 3518 3519	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.
3520	18.2 HTTP Security
3521 3522 3523 3524	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.
3525 3526 3527 3528	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.
3529 3530	Implementations of XRI Proxies and caching authorities should also take special note of the security recommendations in HTTP/1.1 [RFC2616] section 15.7.
3531	18.3 SAML Considerations
3532 3533 3534	SAML trusted authority resolution must adhere to the rules defined by the SAML 2.0 Core Specification <b>[SAML]</b> . Particularly noteworthy are the XML Transform restrictions on XML Signature and the enforcement of the SAML Conditions element regarding the validity period.
3535	18.4 Limitations of Trusted Resolution
3536 3537 3538 3539 3540	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.
3541 3542 3543	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.

# 18.5 Synonym Verification 3545 As discussed in section 5, XRI and XRDS infrastructure has rich support for identifier synonyms, including synonyms that cross security domains. For this reason it is particularly important that identifier authorities, including registries, registrars, directory administrators, identity providers,

# 18.6 Redirect and Ref Management

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

and other parties who issue XRIs and manage XRDS documents, enforce the security policies

highlighted in section 5 regarding registration and management of XRDS synonym elements.

3555 delegation policies.

3548

3549

3550

3556

3564

3569

## **18.7 Community Root Authorities**

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

## 18.8 Caching Authorities

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

## 18.9 Recursing and Proxy Resolution

- During recursing resolution, subsegments of the XRI authority component for which the resolving network endpoint is not authoritative may be revealed to that service endpoint. During proxy resolution, some or all of an XRI is provided to the proxy resolver.
- 3573 In both cases, privacy considerations should be evaluated before disclosing such information.

#### 3574 18.10 Denial-Of-Service Attacks

3575 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks typical of systems relying on DNS and HTTP(S).

# A. Acknowledgments

- The editors would like to thank the following current and former members of the OASIS XRI TC for their particular contributions to this and previous versions of this specification:
- William Barnhill, Booz Allen and Hamilton
- Dave McAlpin, Epok
- Chetan Sabnis, Epok
- Peter Davis, Neustar
- Victor Grey, PlaNetwork
- Mike Lindelsee, Visa International
- Markus Sabadello, XDI.org
- 3587 John Bradley

- 3588 Kermit Snelson
- The editors would also like to acknowledge the contributions of the other members of the OASIS XRI Technical Committee, whose other voting members at the time of publication were:
- Geoffrey Strongin, Advanced Micro Devices
- Ajay Madhok, AmSoft Systems
- Dr. XiaoDong Lee, China Internet Network Information
- Nat Sakimura, Nomura Research
- 3595 Owen Davis, PlaNetwork
- 3596 Fen Labalme, PlaNetwork
- Marty Schleiff, The Boeing Company
- Dave Wentker, Visa International
- Paul Trevithick
- The editors also would like to acknowledge the following people for their contributions to previous versions of OASIS XRI specifications (affiliations listed for OASIS members):
- 3602 Marc Le Maitre, Cordance Corporation: Thomas Bikeev, EAN International: Krishna Sankar,
- 3603 Cisco; Winston Bumpus, Dell; Joseph Moeller, EDS; Steve Green, Epok; Lance Hood, Jerry
- 3604 Kindall, Adarbad Master, Davis McPherson, Chetan Sabnis, and Loren West, Epok; Phillipe
- 3605 LeBlanc, Jim Schreckengast, and Xavier Serret, Gemplus; John McGarvey, IBM; Reva Modi,
- 3606 Infosys; Krishnan Rajagopalan, Novell; Masaki Nishitani, Tomonori Seki, and Tetsu Watanabe,
- 3607 Nomura Research Institute; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,
- 3608 Verisign; Rajeev Maria, Terence Spielman, and John Veizades, Visa International; Lark Allen and
- 3609 Michael Willett, Wave Systems; Matthew Dovey; Eamonn Neylon; Mary Nishikawa; Lars Marius
- 3610 Garshol; Norman Paskin; and Bernard Vatant.

## B. RelaxNG Schema for XRDS and XRD

- Following are the locations of the normative RelaxNG compact schema files for XRDS and XRD as defined by this specification:
- xrds.rnc: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrds.mc
- xrd.rnc: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrd.mc
- 3616 IMPORTANT: The **xrd.rnc** schema does NOT include deprecated attribute values that are recommended for backwards compatibility. See the highlighted Backwards Compatibility notes in sections 9.1.1 and 13.3.2 for more details.
- 3619 Listings of these files are provided in this appendix for reference but are non-normative.

#### xrds.rnc

3611

```
3621
        namespace xrds = "xri://$xrds"
3622
        namespace xrd = "xri://$xrd*($v*2.0)"
3623
        namespace local = ""
3624
3625
3626
        datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
        any.element =
3627
          element * {
3628
            ( attribute* { text }*
3629
               text
3630
              any.element )*
3631
3632
          }
3633
        any.external.element =
3634
          element * - ( xrd:XRD | xrds:XRDS ) {
3635
            ( attribute * { text } *
3636
               text
3637
               any.element )*
3638
          }
3639
3640
        other.attribute = attribute * - ( local:* ) { text }
3641
3642
        start = XRDS
3643
3644
        XRDS = element xrds:XRDS {
3645
            other.attribute *,
3646
            ( attribute ref { xs:anyURI } | attribute redirect { xs:anyURI } )?,
3647
            ( any.external.element | XRDS | external "xrd.rnc" )
3648
3649
3650
        xrd.rnc
3651
        default namespace = "xri://$xrd*($v*2.0)"
3652
3653
        namespace xrd = "xri://$xrd*($v*2.0)"
        namespace saml = "urn:oasis:names:tc:SAML:2.0:assertion"
3654
        namespace ds = "http://www.w3.org/2000/09/xmldsig#"
3655
        namespace local = "'
3656
3657
        datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
3658
3659
        start = XRD
3660
3661
        anyelementbody =
3662
            (attribute * {text}
3663
3664
             | element * {anyelementbody} )*
3665
3666
```

```
3667
         non.xrd.element = element * - xrd:* {
3668
             anyelementbody
3669
3670
3671
3672
         other.attribute = attribute * - ( local:* | xrd:* ) { text }
3673
         XRD = element XRD {
3674
             other.attribute *,
3675
             attribute idref { xs:IDREF } ?,
3676
             attribute version { "2.0" } ?,
3677
             XRDType *,
3678
             Query ?,
3679
             Status ?,
3680
             ServerStatus ?,
3681
             Expires ?,
3682
             ProviderID ?,
3683
             ( Redirect+ | Ref+ ) ?,
3684
             LocalID *,
3685
             EquivID *,
3686
             CanonicalID ?,
3687
             CanonicalEquivID ?,
3688
             Service *,
3689
             element saml:Assertion { anyelementbody } ?,
3690
             non.xrd.element *
3691
         }
3692
3693
         XRDType = element Type {
3694
             other.attribute *,
3695
             xs:anyURI
3696
3697
3698
         Query = element Query {
3699
            other.attribute *,
3700
             text
3701
3702
3703
3704
         append.attribute =
             attribute append { "none" | "local" | "authority" | "path" | "query" | "qxri" }
3705
3706
3707
         Status = element Status {
             other.attribute *, attribute code { xs:integer },
3708
             attribute cid { "absent" | "off" | "verified" | "failed" } ?, attribute ceid { "absent" | "off" | "verified" | "failed" } ?,
3709
3710
3711
3712
         }
3713
3714
         ServerStatus = element ServerStatus {
3715
             other.attribute *,
3716
             attribute code { xs:integer },
3717
3718
             text
3719
3720
3721
3722
         Expires = element Expires {
             other.attribute *,
             xs:dateTime
3723
3724
3725
         ProviderID = element ProviderID {
3726
             other.attribute *,
3727
             xs:anyURI
3728
3729
3730
3731
3732
3733
         Redirect = element Redirect {
             other.attribute *,
             attribute priority { xs:integer }?,
             append.attribute ?,
3734
3735
             xs:anyURI
         }
3736
3737
```

```
3738
        Ref = element Ref {
3739
3740
             other.attribute *,
             attribute priority { xs:integer }?,
3741
             xs:anyURI
3742
         }
3743
3744
         LocalID = element LocalID {
3745
             other.attribute *,
3746
             attribute priority { xs:integer } ?,
3747
3748
             xs:anyURI
3749
3750
         EquivID = element EquivID {
3750
3751
3752
3753
3754
3755
3756
             other.attribute *,
             attribute priority { xs:integer } ?,
             xs:anyURI
         CanonicalID = element CanonicalID {
3757
3758
             other.attribute *,
             xs:anyURI
3759
         }
3760
3761
         CanonicalEquivID = element CanonicalEquivID {
3762
             other.attribute *,
3763
             xs:anyURI
3764
3765
3766
         Service = element Service {
3767
3768
             other.attribute *,
             attribute priority { xs:integer }?,
3769
             ProviderID?,
3770
             ServiceType *,
3771
             Path *,
3772
             MediaType *,
3773
             ( URI+ | Redirect+ | Ref+ )?,
3774
3775
             LocalID *,
             element ds:KeyInfo { anyelementbody }?,
3776
             non.xrd.element *
3777
         }
3778
3779
         URI = element URI {
3780
             other.attribute *,
3781
3782
             attribute priority { xs:integer }?,
             append.attribute ?,
3783
             xs:anyURI
3784
3785
3786
         selection.attributes = attribute match { "any" | "default" | "non-null" | "null" } ?,
3787
                                  attribute select { xs:boolean } ?
3788
3789
3790
         ServiceType = element Type {
             other.attribute *,
3791
3792
3793
             selection.attributes,
             xs:anyURI
3794
3795
3796
         Path = element Path {
             other.attribute *,
3797
             selection.attributes,
3798
             xs:string
3799
3800
3801
         MediaType = element MediaType {
3802
             other.attribute *,
3803
             selection.attributes,
3804
             xs:string
3805
```

## C. XML Schema for XRDS and XRD

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

- xrds.xsd: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrds.xsd
- xrd.xsd: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrd.xsd

3812 IMPORTANT: The **xrd.xsd** schema does NOT include deprecated attribute values that are recommended for backwards compatibility. See the highlighted Backwards Compatibility notes in sections 9.1.1 and 13.3.2 for more details.

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

#### xrds.xsd

3806

3810

3815

3816

```
3817
        <?xml version="1.0" encoding="UTF-8"?>
3818
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds"</pre>
3819
        targetNamespace="xri://$xrds" elementFormDefault="qualified">
3820
            <!-- Utility patterns -->
3821
            <xs:attributeGroup name="otherattribute">
3822
                <xs:anyAttribute namespace="##other" processContents="lax"/>
3823
            </xs:attributeGroup>
3824
            <xs:group name="otherelement">
3825
                <xs:choice>
3826
                   <xs:any namespace="##other" processContents="lax"/>
3827
                    <xs:any namespace="##local" processContents="lax"/>
3828
3829
            </xs:aroup>
3830
            <!-- Patterns for elements -->
3831
            <xs:element name="XRDS">
3832
                <xs:complexType>
3833
                   <xs:sequence>
3834
                       <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3835
                    </xs:sequence>
3836
                    <xs:attributeGroup ref="xrds:otherattribute"/>
3837
                    <!--XML Schema does not currently offer a means to express that only one of
3838
        the following two attributes may be used in any XRDS element, i.e., an XRDS document may
3839
        describe EITHER a redirect identifier or a ref identifier but not both.-->
3840
                    <xs:attribute name="redirect" type="xs:anyURI" use="optional"/>
3841
                    <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
3842
                </xs:complexType>
3843
            </xs:element>
3844
        </xs:schema>
3845
3846
```

#### xrd.xsd

3847

3848

3849

3850

3851

3852

3853

3854

3855

3856

3857

3858

3859 3860

3861

```
3863
            <xs:attributeGroup name="priorityAttrGrp">
3864
                <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
3865
            </xs:attributeGroup>
3866
            <xs:attributeGroup name="codeAttrGrp">
3867
                <xs:attribute name="code" type="xs:int" use="required"/>
3868
            </xs:attributeGroup>
3869
            <xs:attributeGroup name="verifyAttrGrp">
3870
                <xs:attribute name="cid" use="optional">
3871
                   <xs:simpleType>
3872
                       <xs:restriction base="xs:string">
3873
                           <xs:enumeration value="absent"/>
3874
                           <xs:enumeration value="off"/>
3875
                           <xs:enumeration value="verified"/>
3876
                           <xs:enumeration value="failed"/>
3877
                       </xs:restriction>
3878
                    </xs:simpleType>
3879
                </xs:attribute>
3880
                <xs:attribute name="ceid" use="optional">
3881
                   <xs:simpleType>
3882
                       <xs:restriction base="xs:string">
3883
                           <xs:enumeration value="absent"/>
3884
                           <xs:enumeration value="off"/>
3885
                           <xs:enumeration value="verified"/>
3886
                           <xs:enumeration value="failed"/>
3887
                       </xs:restriction>
3888
                    </xs:simpleType>
3889
                </xs:attribute>
3890
            </xs:attributeGroup>
3891
            <xs:attributeGroup name="selectionAttrGrp">
3892
                <xs:attribute name="match" use="optional" default="default">
3893
                    <xs:simpleType>
3894
                       <xs:restriction base="xs:string">
3895
                           <xs:enumeration value="default"/>
3896
                           <xs:enumeration value="any"/>
                           <xs:enumeration value="non-null"/>
3897
3898
                           <xs:enumeration value="null"/>
3899
                       </xs:restriction>
3900
                    </xs:simpleType>
3901
                </xs:attribute>
3902
                <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
3903
            </xs:attributeGroup>
3904
            <xs:attributeGroup name="appendAttrGrp">
3905
                <xs:attribute name="append" use="optional" default="none">
3906
                    <xs:simpleType>
3907
                       <xs:restriction base="xs:string">
3908
                           <xs:enumeration value="none"/>
3909
                           <xs:enumeration value="local"/>
3910
                           <xs:enumeration value="authority"/>
3911
                           <xs:enumeration value="path"/>
3912
                           <xs:enumeration value="query"/>
3913
                           <xs:enumeration value="qxri"/>
3914
                       </xs:restriction>
3915
                   </xs:simpleType>
3916
                </xs:attribute>
3917
            </xs:attributeGroup>
3918
            <xs:complexType name="URIPattern">
3919
                <xs:simpleContent>
3920
                    <xs:extension base="xs:anyURI">
3921
                       <xs:attributeGroup ref="xrd:otherattribute"/>
3922
                   </r></r></r></r>
3923
                </xs:simpleContent>
3924
            </xs:complexType>
3925
            <xs:complexType name="URIPriorityPattern">
3926
                <xs:simpleContent>
3927
                    <xs:extension base="xrd:URIPattern">
3928
                       <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3929
                   </xs:extension>
3930
                </xs:simpleContent>
3931
            </xs:complexType>
3932
```

```
3933
            <xs:complexType name="URIPriorityAppendPattern">
3934
                <xs:simpleContent>
3935
                    <xs:extension base="xrd:URIPriorityPattern">
3936
                       <xs:attributeGroup ref="xrd:appendAttrGrp"/>
3937
                    </xs:extension>
3938
                </xs:simpleContent>
3939
            </xs:complexType>
3940
            <xs:complexType name="StringPattern">
3941
                <xs:simpleContent>
3942
                    <xs:extension base="xs:string">
3943
                       <xs:attributeGroup ref="xrd:otherattribute"/>
3944
                    </xs:extension>
3945
                </xs:simpleContent>
3946
            </xs:complexType>
3947
            <xs:complexType name="StringSelectionPattern">
3948
                <xs:simpleContent>
3949
                    <xs:extension base="xrd:StringPattern">
3950
                       <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
3951
                    </xs:extension>
3952
                </xs:simpleContent>
3953
            </xs:complexType>
3954
            <!-- Patterns for elements -->
3955
            <xs:element name="XRD">
3956
                <xs:complexType>
3957
                   <xs:sequence>
3958
                       <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
3959
                       <xs:element ref="xrd:Query" minOccurs="0"/>
                       <xs:element ref="xrd:Status" minOccurs="0"/>
3960
3961
                       <xs:element ref="xrd:ServerStatus" minOccurs="0"/>
3962
                       <xs:element ref="xrd:Expires" minOccurs="0"/>
3963
                       <xs:element ref="xrd:ProviderID" minOccurs="0"/>
3964
                       <xs:choice>
3965
                           <xs:element ref="xrd:Redirect" minOccurs="0" maxOccurs="unbounded"/>
3966
                           <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
3967
                       </xs:choice>
3968
                        <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
3969
                       <xs:element ref="xrd:EquivID" minOccurs="0" maxOccurs="unbounded"/>
3970
                       <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded"/>
3971
                       <xs:element ref="xrd:CanonicalEquivID" minOccurs="0"</pre>
3972
        maxOccurs="unbounded"/>
3973
3974
                       <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded"/>
                       <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3975
3976
                    <xs:attribute name="idref" type="xs:IDREF" use="optional"/>
3977
                    <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0"/>
3978
                    <xs:attributeGroup ref="xrd:otherattribute"/>
3979
                </xs:complexType>
3980
            </xs:element>
3981
            <xs:element name="Type">
3982
                <xs:complexType>
3983
                    <!--XML Schema does not offer a means to express that usage of the following
3984
        group of optional attributes is only defined when the Type element is used in the context
3985
        of the xrd:XRD/xrd:Service element, and not when it is used in the context of the xrd:XRD
3986
        element.-->
3987
                    <xs:simpleContent>
3988
                       <xs:extension base="xrd:URIPattern">
3989
                           <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
3990
                       </xs:extension>
3991
                    </xs:simpleContent>
3992
                </xs:complexType>
3993
            </xs:element>
3994
            <xs:element name="Query" type="xrd:StringPattern"/>
3995
            <xs:element name="Status">
3996
                <xs:complexType>
3997
                    <xs:simpleContent>
3998
                       <xs:extension base="xrd:StringPattern">
3999
                           <xs:attributeGroup ref="xrd:codeAttrGrp"/>
4000
                           <xs:attributeGroup ref="xrd:verifyAttrGrp"/>
4001
                           <xs:attributeGroup ref="xrd:otherattribute"/>
4002
                       </xs:extension>
4003
                    </xs:simpleContent>
```

```
4004
                </xs:complexType>
4005
            </xs:element>
4006
            <xs:element name="ServerStatus">
4007
                <xs:complexType>
4008
                    <xs:simpleContent>
4009
                        <xs:extension base="xrd:StringPattern">
4010
                            <xs:attributeGroup ref="xrd:codeAttrGrp"/>
4011
                            <xs:attributeGroup ref="xrd:otherattribute"/>
4012
                        </xs:extension>
4013
                    </xs:simpleContent>
4014
                </xs:complexType>
4015
            </xs:element>
4016
            <xs:element name="Expires">
4017
                <xs:complexType>
4018
                    <xs:simpleContent>
4019
                        <xs:extension base="xs:dateTime">
4020
                            <xs:attributeGroup ref="xrd:otherattribute"/>
4021
                        </xs:extension>
4022
                    </xs:simpleContent>
4023
                </xs:complexType>
4024
            </rs:element>
4025
            <xs:element name="ProviderID" type="xrd:URIPattern"/>
4026
            <xs:element name="Redirect" type="xrd:URIPriorityAppendPattern"/>
4027
            <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
4028
            <xs:element name="LocalID">
4029
                <xs:complexType>
4030
                    <xs:simpleContent>
4031
                        <xs:extension base="xrd:StringPattern">
4032
                            <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
4033
                        </xs:extension>
4034
                    </xs:simpleContent>
4035
                </xs:complexType>
4036
            </xs:element>
4037
            <xs:element name="EquivID" type="xrd:URIPriorityPattern"/>
4038
            <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>
4039
            <xs:element name="CanonicalEquivID" type="xrd:URIPriorityPattern"/>
4040
            <xs:element name="Service">
4041
                <xs:complexType>
4042
                    <xs:sequence>
4043
                        <xs:element ref="xrd:ProviderID" minOccurs="0"/>
4044
                        <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
4045
4046
                        <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
4047
                        <xs:choice>
4048
                            <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
4049
                            <xs:element ref="xrd:Redirect" minOccurs="0" maxOccurs="unbounded"/>
4050
                            <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
4051
                        </xs:choice>
4052
                        <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
4053
                        <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
4054
                    </xs:sequence>
4055
                    <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
4056
                    <xs:attributeGroup ref="xrd:otherattribute"/>
4057
                </xs:complexType>
4058
            </xs:element>
4059
            <xs:element name="Path" type="xrd:StringSelectionPattern"/>
4060
            <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
4061
            <xs:element name="URI" type="xrd:URIPriorityAppendPattern"/>
4062
        </xs:schema>
4063
```

#### D. Media Type Definition for application/xrds+xml 4064 This section is prepared in anticipation of filing a media type registration meeting the 4065 4066 requirements of [RFC4288]. 4067 Type name: application 4068 Subtype name: xrds+xml 4069 Required parameters: None 4070 Optional parameters: See Table 6 of this document. 4071 Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], 4072 Section 3.2. 4073 Security considerations: As defined in this specification. In addition, as this media type uses the 4074 "+xml" convention, it shares the same security considerations as described in [RFC3023], 4075 Section 10. 4076 Interoperability considerations: There are no known interoperability issues. 4077 Published specification: This specification. 4078 Applications that use this media type: Applications conforming to this specification use this 4079 media type. 4080 Person & email address to contact for further information: Drummond Reed, OASIS XRI 4081 Technical Committee Co-Chair, drummond.reed@cordance.net 4082 Intended usage: COMMON 4083 Restrictions on usage: None

4084

4085

Author: OASIS XRI TC

Change controller: OASIS XRI TC

#### E. Media Type Definition for application/xrd+xml 4086 4087 This section is prepared in anticipation of filing a media type registration meeting the 4088 requirements of [RFC4288]. 4089 Type name: application 4090 Subtype name: xrd+xml 4091 Required parameters: None 4092 Optional parameters: See Table 6 of this document. 4093 Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], 4094 Section 3.2. Security considerations: As defined in this specification. In addition, as this media type uses the 4095 4096 "+xml" convention, it shares the same security considerations as described in [RFC3023], 4097 Section 10. 4098 Interoperability considerations: There are no known interoperability issues. 4099 Published specification: This specification. 4100 Applications that use this media type: Applications conforming to this specification use this 4101 media type. 4102 Person & email address to contact for further information: Drummond Reed, OASIS XRI 4103 Technical Committee Co-Chair, drummond.reed@cordance.net 4104 Intended usage: COMMON 4105 Restrictions on usage: None 4106 Author: OASIS XRI TC 4107 Change controller: OASIS XRI TC

# F. Example Local Resolver Interface Definition

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

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.

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.	

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.

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.	
<pre>nodefault_t, nodefault_p, nodefault_m</pre>	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.	

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.)

	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

4129 Following is the API and descriptions of the five operations.

#### 1. Resolve Authority to XRDS

```
4131 Result resolveAuthToXRDS(
4132 in string QXRI, in Flags flags);
```

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

4140 4141 4142

4139

4130

#### 2. Resolve Authority to XRD

```
4143 Result resolveAuthToXRD(
4144 in string QXRI, in Flags flags);
```

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

4151 4152

4149

4150

#### 3. Resolve Service Endpoint to XRDS

4154

4160

4161 4162

4163

4164 4165

4166

4167

4168

4169 4170

4176

4177

4178

4179

4180

4181

4182

4183 4184 4185

```
4155 Result resolveSEPToXRDS(
4156 in string QXRI, in string sepType,
4157 in string sepMediaType, in Flags flags);
```

- Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13) and outputs the XRDS as specified in section 8.2.1 when the sep subparameter is TRUE.
  - Returns a binding-specific representation of the resolution result which may include, but is not limited to, XRDS output, success/failure code, exceptions and error context.
  - The final XRD in the output XRDS will either contain at least one instance of the requested service endpoint or an error. *IMPORTANT: Although the resolver will perform service selection, the final XRD is NOT filtered when the Resolution Output Format is an XRDS document. Filtering is only performed when the Resolution Output Format is an XRD document (below).*
  - The XRD element(s) in the output XRDS will be signed or not depending on the value of sam1 flag.

#### 4. Resolve Service Endpoint to XRD

```
A171 Result resolveSEPToXRD(
4172 in string QXRI, in string sepType,
4173 in string sepMediaType, in Flags flags);
```

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

#### 4186 5. Resolve Service Endpoint to URI List

4192

4193 4194

4195

4196

```
4187 Result resolveSepToURIList(
4188 in string QXRI, in string sepType,
4189 in string sepMediaType, in Flags flags);
```

- 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.
  - Returns a binding-specific representation of the resolution result which may include, but not limited to, URI-list output, success/failure code, exceptions and error context.
  - If successful, the output URI-list will contain zero or more elements. It is possible that the selected service contains no URI element and it is up to the consuming application to interpret such a result.

# **G. Revision History**

- 4199 Committee Draft 01 of this specification was published in March 2005 and is available at:
- 4200 http://www.oasis-open.org/committees/download.php/11853
- Significant changes were made based on implementation feedback, resulting in a new implementers draft (Working Draft 10) published in March 2006:
- 4203 http://www.oasis-open.org/committees/download.php/17293
- 4204 All revisions since Working Draft 10 have been tracked on the XRI Technical Committee wiki 4205 page for Working Draft 11:
- 4206 http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11
- 4207 A copy of this wiki page as of the date of this specification has been archived at:
- http://www.oasis-open.org/committees/download.php/26277
- 4209 Due to the extent of the revisions from Committee Draft 01, Committee Draft 02 should be 4210 considered a new document.
- 4211 Committee Draft 03 includes the following revisions based on comments received during the 4212 public review of Committee Draft 02:
- The reference to the XRI Syntax 2.0 specification in section 1.5 was updated.
- The XRD elements in sections 4.2.1 4.2.6 were reformatted to include attribute definitions as separate bullet points (per comment received from Eran Hammer-Lahav).
- The xrd:XRD/xrd:Type element was added to the XRD schema (section 4.2.1 and Appendix B and C) to reuse the xrd:XRD/xrd:Service/xrd:Type element at the XRD level in order to support extension specifications (per comment received from Eran Hammer-Lahav). A reference to this change was added in section 17.1.1
- The flowcharts in Figures 5, 6, 7, and 8 were edited for improved clarity about recording XRDs and nested XRDS documents and clarify using a Redirect URI as an input.
- The Next Authority URI construction algorithm in section 9.1.10 was revised slightly to accommodate using query strings.
- The wording of the bullet points in section 12.1 were clarified (per comment received from 4225 Eran Hammer-Lahav).
- A fourth example was added in section 12.5.1 to illustrate double XRDS nesting.
- Clarifications were made to the pseudocode in section 13.6.
- The CanonicalID verification rule for XRIs was simplified to eliminate the need to involve the xrd:XRD/xrd:ProviderID element (per suggestion from editor William Tan).
- Several typos and incorrect internal references were fixed.
- Several errors were fixed in the RNC schema.