



# Extensible Resource Identifier (XRI) Resolution Version 2.0

Working Draft 11, RC 1

15 November 2007

## Specification URIs:

### This Version:

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

### Previous Version:

N/A

### Latest Version:

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

### Latest Approved Version:

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

### Technical Committee:

OASIS eXtensible Resource Identifier (XRI) TC

### Chairs:

Gabe Wachob, AmSoft <[gabe.wachob@amsoft.net](mailto:gabe.wachob@amsoft.net)>  
Drummond Reed, Cordance <[drummond.reed@cordance.net](mailto:drummond.reed@cordance.net)>

### Editors:

Gabe Wachob, AmSoft <[gabe.wachob@amsoft.net](mailto:gabe.wachob@amsoft.net)>  
Drummond Reed, Cordance <[drummond.reed@cordance.net](mailto:drummond.reed@cordance.net)>  
Les Chasen, NeuStar <[les.chasen@neustar.biz](mailto:les.chasen@neustar.biz)>  
William Tan, NeuStar <[william.tan@neustar.biz](mailto:william.tan@neustar.biz)>  
Steve Churchill, XDI.org <[steven.churchill@xdi.org](mailto: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 1.0, Committee Draft 01, March 2005

**Comment [DSR1]:** TODO-CD:  
These URIs will be adjusted after the Committee Draft vote.

**Comment [DSR2]:** TODO-CD –  
confirm these with Mary McRae

## Declared XML Namespace(s)

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

## Abstract:

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

## Status:

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

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

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

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

---

## Notices

Copyright © OASIS® 1993–2007. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

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

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

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

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

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

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

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

---

# Table of Contents

1	Introduction .....	11
1.1	Overview of XRI Resolution Architecture .....	11
1.2	Structure of this Specification .....	14
1.3	Terminology and Notation .....	15
1.4	Examples .....	16
1.5	Normative References .....	16
1.6	Non-Normative References .....	17
2	Conformance .....	18
2.1	Conformance Targets .....	18
2.2	Conformance Claims .....	18
2.3	XRDS Clients .....	18
2.4	XRDS Servers .....	18
2.5	XRI Local Resolvers .....	19
2.5.1	Generic .....	19
2.5.2	HTTPS .....	19
2.5.3	SAML .....	19
2.6	XRI Proxy Resolvers .....	19
2.6.1	Generic .....	19
2.6.2	HTTPS .....	19
2.6.3	SAML .....	19
2.7	XRI Authority Servers .....	20
2.7.1	Generic .....	20
2.7.2	HTTPS .....	20
2.7.3	SAML .....	20
2.8	Extensions .....	20
2.9	Language .....	20
3	Namespaces .....	21
3.1	XRI Namespaces for XRI Resolution .....	21
3.1.1	XRIs Reserved for XRI Resolution .....	21
3.1.2	XRIs Assigned to XRI Resolution Service Types .....	21
3.2	XML Namespaces for XRI Resolution .....	21
3.3	Media Types for XRI Resolution .....	22
4	XRDS Documents .....	24
4.1	XRDS and XRD Namespaces .....	24
4.2	XRD Elements and Attributes .....	24
4.2.1	Management Elements .....	26
4.2.2	Trust Elements .....	27
4.2.3	Synonym Elements .....	27
4.2.4	Service Endpoint Descriptor Elements .....	28
4.2.5	Service Endpoint Trust Elements .....	29
4.2.6	Service Endpoint Selection Elements .....	29
4.3	XRD Attribute Processing Rules .....	30

4.3.1 ID Attribute .....	30
4.3.2 Version Attribute .....	30
4.3.3 Priority Attribute .....	30
4.4 XRI and IRI Encoding Requirements.....	31
5 XRD Synonym Elements .....	32
5.1 Query Identifiers .....	32
5.1.1 HTTP(S) URI Query Identifiers.....	32
5.1.2 XRI Query Identifiers.....	32
5.2 Synonym Elements .....	33
5.2.1 LocalID.....	33
5.2.2 EquivID .....	33
5.2.3 CanonicalID .....	34
5.2.4 CanonicalEquivID .....	34
5.3 Redirect and Ref Elements .....	35
5.4 XRD Equivalence .....	35
5.5 Synonym Verification.....	36
5.6 Synonym Selection .....	36
6 Discovering an XRDS Document from an HTTP(S) URI.....	37
6.1 Overview .....	37
6.2 HEAD Protocol.....	37
6.3 GET Protocol.....	37
7 XRI Resolution Flow .....	39
8 Inputs and Outputs.....	41
8.1 Inputs .....	41
8.1.1 QXRI (Authority String, Path String, and Query String) .....	43
8.1.2 Resolution Output Format .....	43
8.1.3 Service Type.....	44
8.1.4 Service Media Type .....	45
8.2 Outputs .....	46
8.2.1 XRDS Document.....	48
8.2.2 XRD Element .....	48
8.2.3 URI List.....	49
8.2.4 HTTP(S) Redirect .....	49
9 Generic Authority Resolution Service.....	50
9.1 XRI Authority Resolution .....	50
9.1.1 Service Type and Service Media Type.....	50
9.1.2 Protocol.....	51
9.1.3 Requesting an XRDS Document using HTTP(S) .....	52
9.1.4 Failover Handling.....	54
9.1.5 Community Root Authorities .....	54
9.1.6 Self-Describing XRDS Documents.....	55
9.1.7 Qualified Subsegments .....	56
9.1.8 Cross-References .....	57
9.1.9 Selection of the Next Authority Resolution Service Endpoint .....	57
9.1.10 Construction of the Next Authority URI.....	58

9.1.11	Recurring Authority Resolution .....	58
9.2	IRI Authority Resolution.....	59
9.2.1	Service Type and Media Type .....	59
9.2.2	Protocol.....	59
9.2.3	Optional Use of HTTPS.....	59
10	Trusted Authority Resolution Service .....	60
10.1	HTTPS .....	60
10.1.1	Service Type and Service Media Type .....	60
10.1.2	Protocol.....	60
10.2	SAML .....	60
10.2.1	Service Type and Service Media Type .....	61
10.2.2	Protocol.....	61
10.2.3	Recurring Authority Resolution .....	62
10.2.4	Client Validation of XRDS.....	63
10.2.5	Correlation of ProviderID and KeyInfo Elements .....	64
10.3	HTTPS+SAML .....	64
10.3.1	Service Type and Service Media Type.....	64
10.3.2	Protocol.....	64
11	Proxy Resolution Service .....	66
11.1	Service Type and Media Types .....	66
11.2	HXRIs.....	66
11.3	HXRI Query Parameters .....	68
11.4	HXRI Encoding/Decoding Rules.....	69
11.5	HTTP(S) Accept Headers.....	70
11.6	Null Resolution Output Format .....	71
11.7	Outputs and HTTP(S) Redirects.....	71
11.8	Differences Between Proxy Resolution Servers .....	71
11.9	Combining Authority and Proxy Resolution Servers .....	71
12	Redirect and Ref Processing .....	73
12.1	Cardinality .....	75
12.2	Precedence .....	75
12.3	Redirect Processing .....	76
12.4	Ref Processing.....	77
12.5	Nested XRDS Documents .....	78
12.5.1	Redirect Examples .....	78
12.5.2	Ref Examples.....	80
12.6	Recursion and Backtracking.....	83
13	Service Endpoint Selection .....	85
13.1	Processing Rules .....	85
13.2	Service Endpoint Selection Logic .....	87
13.3	Selection Element Matching Rules.....	88
13.3.1	Selection Element Match Options .....	88
13.3.2	The Match Attribute.....	88
13.3.3	Absent Selection Element Matching Rule .....	89
13.3.4	Empty Selection Element Matching Rule .....	89

13.3.5 Multiple Selection Element Matching Rule .....	89
13.3.6 Type Element Matching Rules .....	89
13.3.7 Path Element Matching Rules .....	89
13.3.8 MediaType Element Matching Rules .....	92
13.4 Service Endpoint Matching Rules .....	92
13.4.1 Service Endpoint Match Options .....	92
13.4.2 Select Attribute Match Rule .....	92
13.4.3 All Positive Match Rule .....	92
13.4.4 Default Match Rule .....	92
13.5 Service Endpoint Selection Rules .....	93
13.5.1 Positive Match Rule .....	93
13.5.2 Default Match Rule .....	93
13.6 Pseudocode .....	93
13.7 Construction of Service Endpoint URIs .....	95
13.7.1 The append Attribute .....	95
13.7.2 The uric Parameter .....	96
14 Synonym Verification .....	97
14.1 Redirect Verification .....	97
14.2 EquivID Verification .....	97
14.3 CanonicalID Verification .....	98
14.3.1 HTTP(S) URI Verification Rules .....	98
14.3.2 XRI Verification Rules .....	99
14.3.3 CanonicalEquivID Verification .....	99
14.3.4 Verification Status Attributes .....	100
14.3.5 Examples .....	100
15 Status Codes and Error Processing .....	105
15.1 Status Elements .....	105
15.2 Status Codes .....	105
15.3 Status Context Strings .....	108
15.4 Returning Errors in Plain Text or HTML .....	108
15.5 Error Handling in Recursing and Proxy Resolution .....	108
16 Use of HTTP(S) .....	109
16.1 HTTP Errors .....	109
16.2 HTTP Headers .....	109
16.2.1 Caching .....	109
16.2.2 Location .....	109
16.2.3 Content-Type .....	109
16.3 Other HTTP Features .....	109
16.4 Caching and Efficiency .....	110
16.4.1 Resolver Caching .....	110
16.4.2 Synonyms .....	110
17 Extensibility and Versioning .....	111
17.1 Extensibility .....	111
17.1.1 Extensibility of XRDS .....	111
17.1.2 Other Points of Extensibility .....	111

17.2	Versioning .....	112
17.2.1	Version Numbering .....	112
17.2.2	Versioning of the XRI Resolution Specification .....	112
17.2.3	Versioning of Protocols .....	112
17.2.4	Versioning of XRDs .....	113
18	Security and Data Protection .....	114
18.1	DNS Spoofing or Poisoning .....	114
18.2	HTTP Security .....	114
18.3	SAML Considerations .....	114
18.4	Limitations of Trusted Resolution .....	114
18.5	Synonym Verification .....	115
18.6	Redirect and Ref Management .....	115
18.7	Community Root Authorities .....	115
18.8	Caching Authorities .....	115
18.9	Recurring and Proxy Resolution .....	115
18.10	Denial-Of-Service Attacks .....	115
A.	Acknowledgments .....	116
B.	RelaxNG Schema for XRDS and XRD .....	117
C.	XML Schema for XRDS and XRD .....	120
D.	Media Type Definition for application/xrds+xml .....	124
E.	Media Type Definition for application/xrd+xml .....	125
F.	Example Local Resolver Interface Definition (Informative) .....	126
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.....	47
Figure 5: Authority resolution flowchart.....	51
Figure 6: XRDS request flowchart.....	53
Figure 7: Redirect and Ref processing flowchart.....	74
Figure 8: Service endpoint selection flowchart.....	85
Figure 9: Service endpoint (SEP) selection logic flowchart.....	87

---

## Table of Tables

Table 1: Comparing DNS and XRI resolution architecture.....	12
Table 2: XRIs reserved for XRI resolution. ....	21
Table 3: XRIs assigned to identify XRI resolution service types. ....	21
Table 4: XML namespace prefixes used in this specification.....	22
Table 5: Media types defined or used in this specification. ....	22
Table 6: Parameters for the media types defined in Table 5.....	23
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.....	46
Table 11: Service Type and Service Media Type values for generic authority resolution. ....	50
Table 12: Parsing the first subsegment of an XRI that begins with a global context symbol.....	57
Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference. ....	57
Table 14: Examples of the Next Authority URIs constructed using different types of cross-references. ...	57
Table 15: Service Type and Service Media Type values for HTTPS trusted authority resolution.....	60
Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.....	61
Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution..	64
Table 18: Service Type and Service Media Type values for proxy resolution. ....	66
Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.....	68
Table 20: Example of HXRI components prior to transformation to URI-normal form. ....	70
Table 21: Example of HXRI components after transformation to URI-normal form.....	70
Table 22: Example of HXRI components after application of the required encoding rules.....	70
Table 23: Comparison of Redirect and Ref elements. ....	73
Table 24: Match options for selection elements.....	88
Table 25: Enumerated values of the global match attribute and corresponding matching rules.....	88
Table 26: Examples of applying the Path element matching rules.....	91
Table 27: Match options for service endpoints. ....	92
Table 28: Values of the <code>append</code> attribute and the corresponding QXRI component to append.....	95
Table 29: Error codes for XRI resolution.....	108

# 1 Introduction

Extensible Resource Identifier (XRI) provides a uniform syntax for abstract structured identifiers as defined in [XRISyntax]. Because XRIs may be used across a wide variety of communities and applications (as Web addresses, messaging addresses, database keys, filenames, directory keys, object IDs, XML IDs, tags, etc.), no single resolution mechanism may prove appropriate for all XRIs. However, in the interest of promoting interoperability, this specification defines a simple generic resource description format called XRDS (Extensible Resource Descriptor Sequence), a standard protocol for 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 that describe a host. For instance, an application might ask a DNS resolver for the A (address) record for docs.oasis-open.org. If the resolver does not have the answer cached, it will start by querying one of the well-known DNS root nameservers for this fully qualified domain name. Since DNS names work from right to left, and since the root nameservers only know about top level domains, it will return the NS (name server) records for the top-level domain org. 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 didn't have the answer cached, it would resolve the domain name and return the results back to the resolver. A recursing nameserver typically caches all these resource records so it can answer subsequent queries directly from cache.

XRI resolution follows this same architecture except at a higher level of abstraction, i.e., rather than using UDP to resolve a domain name into 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.

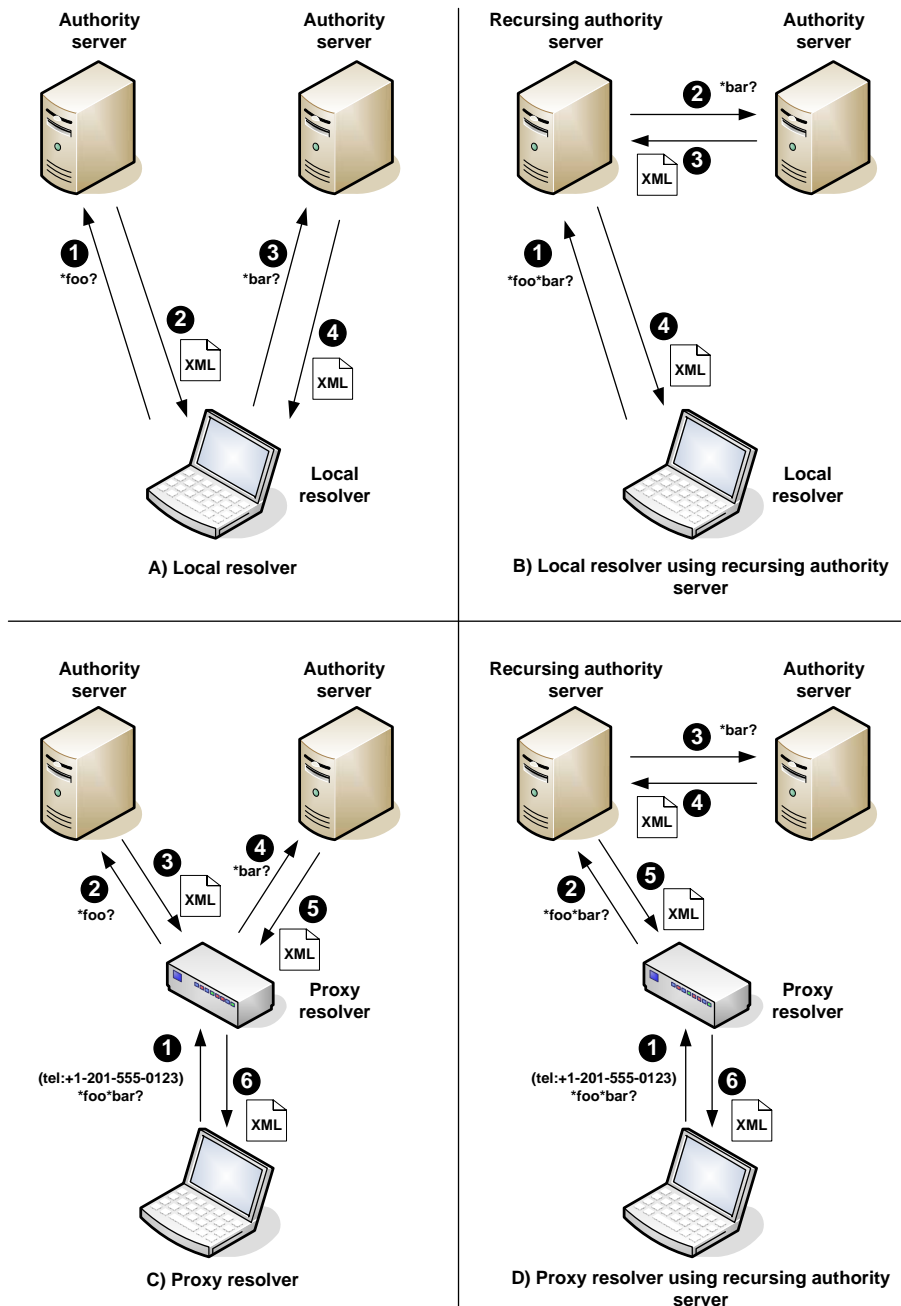
Resolution Component	DNS Architecture	XRI Architecture
Identifier	domain name	XRI (authority + path + query)
Resource record format	text (resource record)	XML (XRDS document)
Attribute identifier	string	anyURI
Network endpoint identifier	IP address	URI
Synonyms	CNAME	LocalID, EquivID, CanonicalID, CanonicalEquivID
Primary resolution protocol	UDP	HTTP(S)
Trusted resolution options	DNSSEC	HTTPS and/or SAML
Resolution client	resolver	resolver

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

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

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

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



42  
 43 Figure 1: Four typical scenarios for XRI authority resolution.

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

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

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

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

## 89 1.2 Structure of this Specification

90 This specification is structured into the following sections:

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

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

95 The next three sections cover XRDS documents and the requirements for XRDS clients and  
96 servers:

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

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

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

### 131 **1.3 Terminology and Notation**

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

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

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

140 Formatting conventions used in this document:

141 Examples look like this.

142 ABNF productions look like this.

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

## 144 1.4 Examples

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

## 150 1.5 Normative References

- 151 **[DNSSEC]** D. Eastlake, *Domain Name System Security Extensions*,  
152 <http://www.ietf.org/rfc/rfc2535>, IETF RFC 2535, March 1999.
- 153 **[RFC2045]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*  
154 *Part One: Format of Internet Message Bodies*,  
155 <http://www.ietf.org/rfc/rfc2045.txt>, IETF RFC 2045, November 1996.
- 156 **[RFC2046]** N. Borenstein, N. Freed, *Multipurpose Internet Mail Extensions (MIME)*  
157 *Part Two: Media Types*, <http://www.ietf.org/rfc/rfc2046.txt>, IETF RFC  
158 2046, November 1996.
- 159 **[RFC2119]** S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*,  
160 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 161 **[RFC2141]** R. Moats, *URN Syntax*, <http://www.ietf.org/rfc/rfc2141.txt>, IETF RFC  
162 2141, May 1997.
- 163 **[RFC2483]** M. Mealling, R. Daniel Jr., *URI Resolution Services Necessary for URN*  
164 *Resolution*, <http://www.ietf.org/rfc/rfc2483.txt>, IETF RFC 2483, January  
165 1999.
- 166 **[RFC2616]** R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.  
167 Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*,  
168 <http://www.ietf.org/rfc/rfc2616.txt>, IETF RFC 2616, June 1999.
- 169 **[RFC2818]** E. Rescorla, *HTTP over TLS*, <http://www.ietf.org/rfc/rfc2818.txt>, IETF  
170 RFC 2818, May 2000.
- 171 **[RFC3023]** M. Murata, S. St-Laurent, D. Kohn, *XML Media Types*,  
172 <http://www.ietf.org/rfc/rfc3023.txt>, IETF RFC 3023, January 2001.
- 173 **[RFC3986]** T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier*  
174 *(URI): Generic Syntax*, <http://www.ietf.org/rfc/rfc3986.txt>, IETF RFC  
175 3986, January 2005.
- 176 **[RFC4234]** D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications:*  
177 *ABNF*, <http://www.ietf.org/rfc/rfc4234.txt>, IETF RFC 4234, October 2005.
- 178 **[RFC4288]** N. Freed, J. Klensin, *Media Type Specifications and Registration*  
179 *Procedures*, <http://www.ietf.org/rfc/rfc4288.txt>, IETF RFC 4288,  
180 December 2005.
- 181 **[SAML]** S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for*  
182 *the OASIS Security Assertion Markup Language (SAML) V2.0*,  
183 <http://www.oasis-open.org/committees/security>, March 2005.

184	<b>[Unicode]</b>	The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined
185		by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley,
186		2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1
187		( <a href="http://www.unicode.org/versions/Unicode4.0.1">http://www.unicode.org/versions/Unicode4.0.1</a> ) and by Unicode 4.1.0
188		( <a href="http://www.unicode.org/versions/Unicode4.1.0">http://www.unicode.org/versions/Unicode4.1.0</a> ), March, 2005.
189	<b>[UUID]</b>	Open Systems Interconnection – <i>Remote Procedure Call</i> , ISO/IEC
190		11578:1996, <a href="http://www.iso.org/">http://www.iso.org/</a> , August 2001.
191	<b>[XML]</b>	T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau,
192		<i>Extensible Markup Language (XML) 1.0, Third Edition</i> , World Wide Web
193		Consortium, <a href="http://www.w3.org/TR/REC-xml/">http://www.w3.org/TR/REC-xml/</a> , February 2004.
194	<b>[XMLDSig]</b>	D. Eastlake, J. Reagle, D. Solo et al., <i>XML-Signature Syntax and</i>
195		<i>Processing</i> , World Wide Web Consortium,
196		<a href="http://www.w3.org/TR/xmlsig-core/">http://www.w3.org/TR/xmlsig-core/</a> , February, 2002.
197	<b>[XMLID]</b>	J. Marsh, D. Veillard, N. Walsh, <i>xml:id Version 1.0</i> , World Wide Web
198		Consortium, <a href="http://www.w3.org/TR/2005/REC-xml-id-20050909">http://www.w3.org/TR/2005/REC-xml-id-20050909</a> ,
199		September 2005.
200	<b>[XMLSchema]</b>	H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, <i>XML Schema Part</i>
201		<i>1: Structures Second Edition</i> , World Wide Web Consortium,
202		<a href="http://www.w3.org/TR/xmlschema-1/">http://www.w3.org/TR/xmlschema-1/</a> , October 2004.
203	<b>[XMLSchema2]</b>	P. Biron, A. Malhotra, <i>XML Schema Part 2: Datatypes Second Edition</i> ,
204		World Wide Web Consortium, <a href="http://www.w3.org/TR/xmlschema-2/">http://www.w3.org/TR/xmlschema-2/</a> ,
205		October 2004.
206	<b>[XRIMetadata]</b>	D. Reed, <i>Extensible Resource Identifier (XRI) Metadata V2.0</i> ,
207		<a href="http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf">http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf</a> ,
208		March 2005.
209	<b>[XRISyntax]</b>	D. Reed, D. McAlpin, <i>Extensible Resource Identifier (XRI) Syntax V2.0</i> ,
210		<a href="http://docs.oasis-open.org/xri/xri/V2.0/xri-syntax-V2.0-cd-01.pdf">http://docs.oasis-open.org/xri/xri/V2.0/xri-syntax-V2.0-cd-01.pdf</a> , March
211		2005.

## 212 1.6 Non-Normative References

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

---

## 223 2 Conformance

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

### 226 2.1 Conformance Targets

227 The conformance targets of this specification are:

- 228 1. *XRDS clients*, which provide a limited subset of the functionality of XRI resolvers.
- 229 2. *XRDS servers*, which provide a limited subset of the functionality of XRI authority servers.
- 230 3. *XRI local resolvers*, which may implement any combination of the generic, HTTPS, or  
231 SAML resolution protocols.
- 232 4. *XRI proxy resolvers*, which may implement any combination of the generic, HTTPS, or  
233 SAML resolution protocols.
- 234 5. *XRI authority servers*, which may implement any combination of the generic, HTTPS, or  
235 SAML resolution protocols.

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

### 239 2.2 Conformance Claims

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

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

### 245 2.3 XRDS Clients

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

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

### 251 2.4 XRDS Servers

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

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

## 257 **2.5 XRI Local Resolvers**

### 258 **2.5.1 Generic**

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

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

### 270 **2.5.2 HTTPS**

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

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

### 275 **2.5.3 SAML**

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

- 278 1. It implements the resolver requirements of the SAML trusted resolution protocol specified  
279 in section 10.2.
- 280 2. It SHOULD also meet the requirements of an HTTPS local resolver. This is STRONGLY  
281 RECOMMENDED for confidentiality of SAML interactions.

## 282 **2.6 XRI Proxy Resolvers**

### 283 **2.6.1 Generic**

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

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

### 287 **2.6.2 HTTPS**

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

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

### 291 **2.6.3 SAML**

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

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

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

## 297 **2.7 XRI Authority Servers**

### 298 **2.7.1 Generic**

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

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

### 308 **2.7.2 HTTPS**

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

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

### 313 **2.7.3 SAML**

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

- 316 1. It implements the server requirements of the SAML trusted resolution protocol specified  
317 in section 10.2.
- 318 2. It SHOULD also meet the requirements of an HTTPS authority server. This is  
319 STRONGLY RECOMMENDED for confidentiality of SAML interactions.

## 320 **2.8 Extensions**

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

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

## 325 **2.9 Language**

326 This specification's normative language is English. Translation into other languages is  
327 encouraged.

## 328 3 Namespaces

### 329 3.1 XRI Namespaces for XRI Resolution

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

#### 334 3.1.1 XRIs Reserved for XRI Resolution

335 The XRIs in Table 2 are assigned by this specification for the purposes of XRI resolution and  
336 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

337 Table 2: XRIs reserved for XRI resolution.

#### 338 3.1.2 XRIs Assigned to XRI Resolution Service Types

339 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

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

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

### 344 3.2 XML Namespaces for XRI Resolution

345 Throughout this document, the following XML namespace prefixes have the meanings defined in  
346 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

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

### 348 3.3 Media Types for XRI Resolution

349 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as  
 350 defined by [RFC2046], particularly in HTTP Accept headers as specified in [RFC2616]. Table 5  
 351 specifies the media types used for XRI resolution. Note that in XRI authority resolution, these  
 352 media types MUST be passed as HTTP Accept header values. By contrast, in XRI proxy  
 353 resolution these media types MUST be passed as query parameters in an HTTP(S) URI as  
 354 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]

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

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

Media Type Parameter	Default Value	Usage	See Section
https	FALSE	Specifies use of HTTPS trusted resolution	10.1
saml	FALSE	Specifies use of SAML trusted resolution	10.2
refs	TRUE	Specifies whether Refs should be followed during resolution (by default they are followed)	12.4
sep	FALSE	Specifies whether service endpoint selection should be performed	13
nodefault_t	TRUE	Specifies whether a default match on a Type service	13.2

		endpoint selection element is allowed	
nodefault_p	TRUE	Specifies whether a default match on a Path service endpoint selection element is allowed	13.2
nodefault_m	TRUE	Specifies whether a default match on a MediaType service endpoint selection element is allowed	13.2
uric	FALSE	Specifies whether a resolver should automatically construct service endpoint URIs	13.6.1
cid	TRUE	Specifies whether automatic canonical ID verification should performed	14.3

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

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

363

## 4 XRDS Documents

364  
365  
366  
367  
368

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

369

### 4.1 XRDS and XRD Namespaces

370  
371  
372  
373  
374  
375

An XRDS document is intended to serve exclusively as an XML container document for XML schemas from other XML namespaces. Therefore it has only a single root element `xrds:XRDS` in its own XML namespace identified by the XRI `xri://$xrds`. It also has two attributes, `redirect` and `ref`, that are used to identify the resource described by the XRDS document. Both are of type `anyURI`. Use of these attributes is defined in section 12.5. A link to the formal RelaxNG schema definition of an XRDS document is provided in Appendix B.

376  
377  
378  
379  
380

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

381  
382

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

383  
384  
385  
386

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

387

### 4.2 XRD Elements and Attributes

388  
389  
390  
391

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

392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410

```
<XRDS xmlns="xri://$xrds" ref="xri://(tel:+1-201-555-0123)*foo">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
    <Query>*foo</Query>
    <Status code="100"/>
    <ServerStatus code="100"/>
    <Expires>2005-05-30T09:30:10Z</Expires>
    <ProviderID>xri://(tel:+1-201-555-0123)</ProviderID>
    <LocalID>*baz</LocalID>
    <EquipID>https://example.com/example/resource/</EquipID>
    <CanonicalID>xri://(tel:+1-201-555-0123)!1234</CanonicalID>
    <CanonicalEquipID>
      xri://=!4a76!c2f7!9033.78bd
    </CanonicalEquipID>
    <Service>
      <ProviderID>
        xri://(tel:+1-201-555-0123)!1234
      </ProviderID>
      <Type>xri://$res*auth*($v*2.0)</Type>
      <MediaType>application/xrds+xml</MediaType>
```

```

411 <URI priority="10">http://resolve.example.com</URI>
412 <URI priority="15">http://resolve2.example.com</URI>
413 <URI>https://resolve.example.com</URI>
414 </Service>
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;https=true</MediaType>
421 <URI>https://resolve.example.com</URI>
422 </Service>
423 <Service>
424 <Type match="null" />
425 <Path select="true">/media/pictures</Path>
426 <MediaType select="true">image/jpeg</MediaType>
427 <URI append="path" >http://pictures.example.com</URI>
428 </Service>
429 <Service>
430 <Type match="null" />
431 <Path select="true">/media/videos</Path>
432 <MediaType select="true">video/mpeg</MediaType>
433 <URI append="path" >http://videos.example.com</URI>
434 </Service>
435 <Service>
436 <ProviderID> xri://!!1000!1234.5678</ProviderID>
437 <Type match="null" />
438 <Path match="default" />
439 <URI>http://example.com/local</URI>
440 </Service>
441 <Service>
442 <Type>http://example.com/some/service/v3.1</Type>
443 <URI>http://example.com/some/service/endpoint</URI>
444 <LocalID>https://example.com/example/resource</LocalID>
445 </Service>
446 </XRD>
447 </XRDS>

```

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

## 451 4.2.1 Management Elements

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

### 456 **xrd:XRD**

457 Container element for all other XRD elements. Implicitly includes an OPTIONAL `xml:id`  
458 attribute of type `xs:ID`. This attribute is REQUIRED in trusted resolution to uniquely  
459 identify this element within the containing `xrds:XRDS` document. It also includes an  
460 OPTIONAL `idref` attribute of type `xs:idref`. This attribute is REQUIRED in trusted  
461 resolution when an XRD element in a nested `xrd:XRDS` document must reference a  
462 previously included XRD instance. See sections 4.3.1 and 12.5. Lastly, it includes a  
463 `version` attribute that is OPTIONAL for uses outside of XRI resolution but REQUIRED  
464 for XRI resolution as defined in section 4.3.2

### 465 **xrd:XRD/xrd:Query**

466 0 or 1 per `xrd:XRD` element. Expresses the XRI, IRI, or URI reference in URI-normal  
467 form whose resolution results in this `xrd:XRD` element. See section 5.1.

### 468 **xrd:XRD/xrd:Status**

469 0 or 1 per `xrd:XRD` element. RECOMMENDED for all XRDs. REQUIRED if the resolver  
470 must report certain error conditions. Contains a REQUIRED attribute `code` of type  
471 `xs:int` that provides a numeric status code. Contains enumerated attributes `cid` and  
472 `ceid` that are OPTIONAL except when REQUIRED to report the results of CanonicalID  
473 verification as defined in section 14.3.4. The contents of the element are a human-  
474 readable message string describing the status of the response as determined by the  
475 resolver. For XRI resolution, values of the Status element and `code` attribute are defined  
476 in section 15.

### 477 **xrd:XRD:xrdServerStatus**

478 0 or 1 per `xrd:XRD` element. Identical to `xrd:XRD/xrd:Status` except this element is  
479 used by an XRI authority server to report the status of a resolution request to an XRI  
480 resolver, and it does not include the `cid` and `ceid` attributes. See section 15.1.

### 481 **xrd:XRD/xrd:Expires**

482 0 or 1 per `xrd:XRD` element. The date/time, in the form of `xs:dateTime`, after which  
483 this XRD cannot be relied upon. To promote interoperability, this date/time value  
484 SHOULD use the UTC "Z" time zone and SHOULD NOT use fractional seconds. A  
485 resolver MUST NOT use an XRD after the time stated here. A resolver MAY discard this  
486 XRD before the time indicated in this result. If the HTTP transport caching semantics  
487 specify an expiry time earlier than the time expressed in this attribute, then a resolver  
488 MUST NOT use this XRD after the expiry time declared in the HTTP headers per section  
489 13.2 of [RFC2616]. See section 16.2.1.

### 490 **xrd:XRD/xrd:Redirect**

491 0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST contain an absolute HTTP(S)  
492 URI. Accepts the optional global `priority` attribute (section 4.3.3). Choice between this  
493 or the `xrd:XRD/xrd:Ref` element below. MUST be processed by a resolver to locate  
494 another XRDS document authorized to describe the target resource as defined in section  
495 12. Includes an optional `append` attribute that governs construction of the final redirect  
496 URI as defined in section 13.6.

### 497 **xrd:XRD/xrd:Ref**

498 0 or more more per `xrd:XRD` element. Type `xs:anyURI`. MUST contain an absolute  
499 XRI. Accepts the optional global `priority` attribute (section 4.3.3). Choice between this  
500 or the `xrd:XRD/xrd:Redirect` element above. MUST be processed by a resolver  
501 (depending on the value of the `refs` subparameter) to locate another XRDS document  
502 authorized to describe the target resource as defined in section 12.

## 503 4.2.2 Trust Elements

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

### 508 `xrd:XRD/xrd:ProviderID`

509 0 or 1 per `xrd:XRD` element. A unique identifier of type `xs:anyURI` for the parent  
510 authority providing this XRD. The value of this element MUST be a persistent identifier.  
511 There MUST be negligible probability that the value of this element will be assigned as an  
512 identifier to any other authority. For purposes of CanonicalID verification (section 14.3), it  
513 is RECOMMENDED to use a fully persistent XRI as defined in [XRISyntax]. If a URN  
514 [RFC2141] or other persistent identifier is used, it is RECOMMENDED to express it as an  
515 XRI cross-reference as defined in [XRISyntax]. Note that for XRI authority resolution, the  
516 authority identified by this element is the parent authority (the provider of the current  
517 XRD), not the child authority (the target of the current XRD). The latter is identified by the  
518 `xrd:XRD/xrd:Service/xrd:ProviderID` element inside a authority resolution  
519 service endpoint (see below).

### 520 `xrd:XRD/saml:Assertion`

521 0 or 1 per `xrd:XRD` element. A SAML assertion from the provider of the current XRD  
522 that asserts that the information contained in the current XRD is authoritative. Because  
523 the assertion is digitally signed and the digital signature encompasses the containing  
524 `xrd:XRD` element, it also provides a mechanism for the recipient to detect unauthorized  
525 changes since the last time the XRD was published.

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

## 531 4.2.3 Synonym Elements

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

### 536 `xrd:XRD/xrd:LocalID`

537 0 or more per `xrd:XRD` element. Type `xs:anyURI`. Accepts the optional global  
538 `xrd:priority` attribute (section 4.3.3). Asserts an interchangeable synonym for the  
539 value of the `xrd:Query` element. See section 5.2.1 for detailed requirements.

### 540 `xrd:XRD/xrd:EquiVID`

541 0 or more per `xrd:XRD` element. Type `xs:anyURI`. Accepts the optional global  
542 `priority` attribute (section 4.3.3). Asserts an absolute identifier for the target resource  
543 that is not equivalent to the CanonicalID or CanonicalEquiVID (see below). See section  
544 5.2.2 for detailed requirements.

545 **xrd:XRD/xrd:CanonicalID**  
546 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical identifier assigned  
547 to the target resource by the authority providing the XRD. See section 5.2.3 for detailed  
548 requirements.

549 **xrd:XRD/xrd:CanonicalEquivID**  
550 0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical identifier for the  
551 target resource assigned by *any* identifier authority. See section 5.2.4 for detailed  
552 requirements.

#### 553 4.2.4 Service Endpoint Descriptor Elements

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

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

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

566 **xrd:XRD/xrd:Service**  
567 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.  
568 Referred to by the abbreviation *SEP*.

569 **xrd:XRD/xrd:Service/xrd:LocalID**  
570 0 or more per `xrd:XRD/xrd:Service` element. Identical to the  
571 `xrd:XRD/xrd:LocalID` element defined above except this synonym is asserted by the  
572 provider of the service and not the parent authority for the XRD. **MAY** be used to provide  
573 one or more identifiers by which the target resource **SHOULD** be identified in the context  
574 of the service endpoint. See section 5.2.1 for detailed requirements.

575 **xrd:XRD/xrd:Service/xrd:URI**  
576 0 more per `xrd:XRD/xrd:Service` element. Type `xs:anyURI`. Choice between this or  
577 the `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`  
578 elements. If present, it indicates a transport-level URI for accessing the capability  
579 described by the parent `Service` element. For the service types defined for XRI resolution  
580 in section 3.1.2, this URI **MUST** be an HTTP or HTTPS URI. Other services may use  
581 other transport protocols. Includes an optional `append` attribute that governs construction  
582 of the final service endpoint URI as defined in section 13.6.

583 **xrd:XRD/xrd:Service/xrd:Redirect**  
584 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the  
585 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Ref` elements.  
586 Identical to the `xrd:XRD/xrd:Redirect` element defined above except processed only  
587 in the context of service endpoint selection. See section 12.

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

589 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the  
590 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Redirect`  
591 elements. Identical to the `xrd:XRD/xrd:Ref` element defined above except processed  
592 only in the context of service endpoint selection. See section 12.

## 593 4.2.5 Service Endpoint Trust Elements

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

### 597 `xrd:XRD/xrd:Service/xrd:ProviderID`

598 0 or 1 per `xrd:XRD/xrd:Service` element. Identical to the  
599 `xrd:XRD/xrd:ProviderID` above, except this identifies the provider of the *described*  
600 *service endpoint* instead of the provider of the XRD. For an XRI authority resolution  
601 service endpoint, it identifies the *child authority* who will perform resolution of subsequent  
602 XRI subsegments. In SAML trusted resolution, when a resolution request is made to the  
603 child authority at this service endpoint, the contents of the `xrd:XRD/xrd:ProviderID`  
604 element in the response MUST match the content of this element for correlation as  
605 defined in section 10.2.5. The same usage MAY apply to other services not defined in  
606 this specification. Authors of other specifications employing XRD service endpoints  
607 SHOULD define the scope and usage of this element, particularly for trust verification.

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

609 0 or 1 per `xrd:XRD/xrd:Service` element. This element provides the digital signature  
610 metadata necessary to validate interaction with the resource identified by the  
611 `xrd:XRD/xrd:Service/xrd:ProviderID` (above). In XRI resolution, this element  
612 comprises the key distribution method for SAML trusted authority resolution as defined in  
613 section 10.2.5. The same usage MAY apply to other services not defined in this  
614 specification.

## 615 4.2.6 Service Endpoint Selection Elements

616 The final set of service endpoint descriptor elements is used in XRI resolution for service endpoint  
617 selection. They include two global attributes used for this purpose: `match` and `select`. See  
618 sections 13.2.2 and 13.3.2.

### 619 `xrd:XRD/xrd:Service/xrd:Type`

620 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI`  
621 that identifies the type of capability available at this service endpoint. See section 3.1.2  
622 for the resolution service types defined in this specification. If a service endpoint does not  
623 include at least one `xrd:Type` element, the service type is effectively described by the  
624 type of URI specified in the `xrd:XRD/xrd:Service/xrd:URI` element, i.e., an HTTP  
625 URI specifies an HTTP service. See section 13.2.6 for Type element matching rules.

### 626 `xrd:XRD/xrd:Service/xrd:Path`

627 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string  
628 meeting the `xri-path` production defined in section 2.2.3 of [XRI Syntax]. See section  
629 13.2.7 for Path element matching rules.

### 630 `xrd:XRD/xrd:Service/xrd:MediaType`

631 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of  
632 content available at this service endpoint. The value of this element MUST be of the form  
633 of a media type defined in [RFC2046]. See section 3.3 for the media types used in XRI  
634 resolution. See section 13.2.8 for MediaType element matching rules.

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

## 639 4.3 XRD Attribute Processing Rules

### 640 4.3.1 ID Attribute

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

Comment [DSR3]: PROOF –  
Added to document why we don't  
declare the `xml:id` attribute.

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

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

### 659 4.3.2 Version Attribute

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

### 666 4.3.3 Priority Attribute

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

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

- 676 1. The consuming application SHOULD select the element instance with the lowest numeric  
677 value of the `priority` attribute. For example, an element with `priority` attribute value  
678 of “10” should be selected before an element with a `priority` attribute value of “11”,  
679 and an element with `priority` attribute value of “11” should be selected before an  
680 element with a `priority` attribute value of “25”. Zero is the highest `priority` attribute

- 681 value. Null is the lowest `priority` attribute value—it is the equivalent of a value of  
682 infinity. It is RECOMMENDED to use a large finite value (100 or more) rather than a null  
683 value.
- 684 2. If an element has no `priority` attribute, its `priority` attribute value is considered to  
685 be null, i.e., the lowest possible priority value. Rather than omitting a `priority` attribute,  
686 it is RECOMMENDED that XRI authorities follow the standard practice in DNS and set  
687 the default `priority` attribute value to “10”.
- 688 3. If two or more instances of the same element type have identical `priority` attribute  
689 values (including the null value), the consuming application SHOULD select one of the  
690 instances at random. This consuming application SHOULD NOT simply choose the first  
691 instance that appears in XML document order.

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

Comment [DSR4]: PROOF – new warning regarding the importance of priority suggested by John.

- 697 4. An element selected according to these rules is referred to in this specification as *the*  
698 *highest priority element*. If this element is subsequently disqualified from the set of  
699 qualified elements, the next element selected according to these rules is referred to as  
700 *the next highest priority element*. If a resolution operation specifying selection of the  
701 highest priority element fails, the resolver SHOULD attempt to select the next highest  
702 priority element unless otherwise specified. This process SHOULD be continued for all  
703 other instances of the qualified elements until success is achieved or all instances are  
704 exhausted.

#### 705 4.4 XRI and IRI Encoding Requirements

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

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

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

719

## 5 XRD Synonym Elements

720  
721  
722

XRDS architecture includes support for *synonyms*—XRI, IRI, or URI 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

723

Table 7: The four XRD synonym elements.

724

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

725

### 5.1 Query Identifiers

726  
727  
728

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:

729  
730  
731

1. A valid absolute HTTP(S) URI that does not contain an XRI.
2. 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.

732

#### 5.1.1 HTTP(S) URI Query Identifiers

733  
734  
735  
736

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

737  
738

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

739

#### 5.1.2 XRI Query Identifiers

740  
741  
742  
743  
744  
745

If the fully-qualified query identifier is an absolute XRI, the XRDS document to which it resolves (via the protocol specified in section 9.1.2) MAY contain multiple XRDs, each XRD corresponding to one subsegment of the authority component of the XRI. Each XRD SHOULD include an `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-qualified query identifier.

746  
747  
748  
749

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

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

## 753 5.2 Synonym Elements

### 754 5.2.1 LocalID

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

758 At the XRD level, the value of the `xrd:XRD/xrd:LocalID` element asserts a synonym that is  
759 interchangeable with the contents of the `xrd:Query` element in the XRD. This means that  
760 resolution of a LocalID from the same parent authority using the same resolution query  
761 parameters as the current query MUST result in an equivalent XRD as defined in section 5.4. It  
762 also means an XRI resolver MAY use it as an alternate key for that XRD in its cache.

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

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

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

Comment [DSR5]: PROOF –  
Replaced the previous wording to  
reference the new section 5.4

### 778 5.2.2 EquivID

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

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

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

- 785 1. When it is REQUIRED as a backpointer to verify another EquivID element in a different  
786 XRD as specified in section 14.2.
- 787 2. When it is REQUIRED as a backpointer to verify a CanonicalEquivID element as  
788 specified in section 14.3.3.

789 SPECIAL SECURITY CONSIDERATIONS: An EquivID synonym SHOULD NOT be trusted  
790 unless it is verified. This function is not performed automatically by XRI resolvers but may be  
791 easily performed by consuming applications using one additional XRI resolution call as specified  
792 in section 14.2. A parent authority SHOULD NOT permit a child authority to edit the EquivID value  
793 in an XRD without authenticating the child authority and verifying that the child authority is  
794 authorized to use this EquivID value. A parent authority SHOULD NOT assert an EquivID

795 element if the identifier authority to whom it points is not authorized to make a CanonicalEquivID  
796 assertion.

### 797 5.2.3 CanonicalID

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

Comment [DSR6]: PROOF –  
Revised per input from John and to  
reference the new section 5.4

- 802 1. It MUST be an identifier for which the parent authority is the final authority. This means  
803 that resolution of a CanonicalID using the same resolution query parameters as the  
804 current query MUST result in an equivalent XRD as defined in section 5.4.
- 805 2. If the CanonicalID is any XRI except a community root authority XRI (section 9.1.3), it  
806 MUST be a direct child of the parent authority's own CanonicalID. (In XRI resolution the  
807 parent authority's CanonicalID is always in the immediately proceeding XRD in the same  
808 XRDS document, not in a nested XRDS document produced as a result of Redirect and  
809 Ref processing as defined in section 12.5.) For example, if the CanonicalID asserted for a  
810 target resource is `@!1!2!3`, then the CanonicalID for the parent authority must be  
811 `@!1!2`. See section 14.3.2 for details.
- 812 3. Once assigned, a parent authority SHOULD NEVER: a) change or reassign a  
813 CanonicalID value, or b) stop asserting a CanonicalID element in an XRD in which it has  
814 been asserted. For this reason, it is STRONGLY RECOMMENDED to use a persistent  
815 identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].

Comment [DSR7]: PROOF –  
Added reference the new section 5.4

Comment [DSR8]: PROOF: Added  
per suggestion from John to clarify  
the previous wording.

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

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

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

### 828 5.2.4 CanonicalEquivID

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

- 832 1. In order for the value of the `xrd:CanonicalEquivID` element to be verified: a) the  
833 XRD in which it appears MUST include a CanonicalID that can be verified as specified in  
834 section 14.2, and b) the XRD to which it resolves MUST meet the rules specified in  
835 section 14.3.3.
- 836 2. For the same reasons as with a CanonicalID, it is STRONGLY RECOMMENDED to use  
837 a persistent identifier such as a persistent XRI [XRISyntax] or a URN [RFC2141].
- 838 3. Although the CanonicalEquivID associated with a CanonicalID MAY change over time, at  
839 any any one point in time, every XRD from the same parent authority that asserts the

840 same CanonicalID value MUST assert the same CanonicalEquivID value if the XRD  
841 includes a CanonicalEquivID element.

Comment [DSR9]: PROOF - Added to help clarify the rules about CanonicalEquivID usage.

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

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

### 853 5.3 Redirect and Ref Elements

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

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

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

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

### 871 5.4 XRD Equivalence

Comment [DSR10]: PROOF: added this section in RC1 to standardize the definition of XRD equivalence.

872 LocalID and CanonicalID synonyms are required resolve to an XRD that is equivalent to the XRD  
873 in which the synonym is asserted. Two XRDs MUST be considered equivalent if they meet the  
874 following rules:

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

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

## 884 5.5 Synonym Verification

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

## 888 5.6 Synonym Selection

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

- 892 • Only select a verified synonym (see above).
- 893 • Select a persistent synonym, particularly if a long term or immutable reference is required. If  
894 a persistent synonym is present, other reassignable synonyms (including the current fully-  
895 qualified query identifier) SHOULD be treated only as temporary identifiers.
- 896 • Select a CanonicalID if present, verified, and persistent. This identifier SHOULD be used  
897 whenever referencing the target resource in the context of the parent authority issuing the  
898 CanonicalID.
- 899 • If possible, *also* select a CanonicalEquivID if present, verified, and persistent. This identifier  
900 SHOULD be used as a reference to the target resource in any context other than that of the  
901 parent authority.
- 902 • When selecting a synonym to use in the context of a specific service endpoint, follow the  
903 recommendations for use of the `xrd:XRD/xrd:Service/xrd:LocalID` element as  
904 specified in section 5.2.1.

---

905 **6 Discovering an XRDS Document from an**  
906 **HTTP(S) URI**

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

911 **6.1 Overview**

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

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

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

920 **6.2 HEAD Protocol**

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

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

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

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

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

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

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

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

938 **6.3 GET Protocol**

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

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

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

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

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

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

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

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

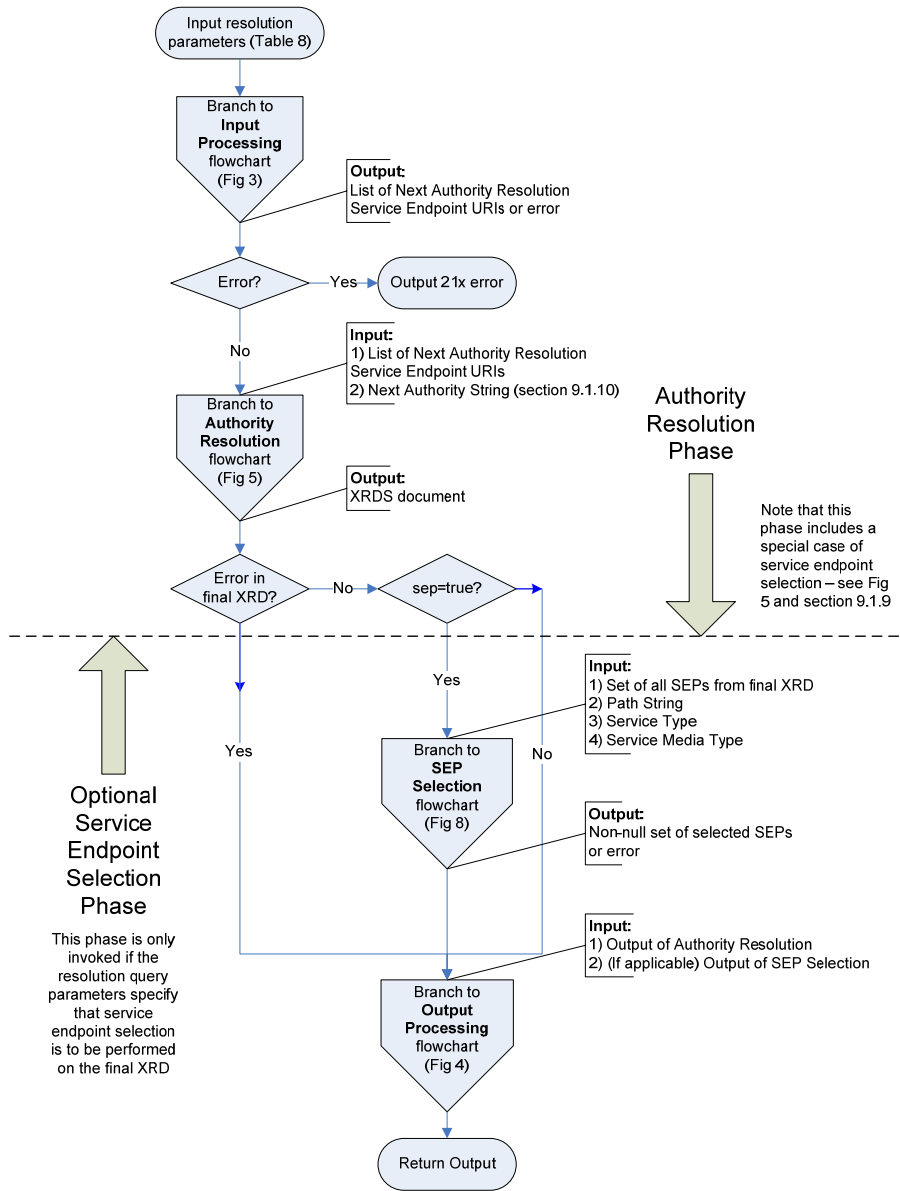
965 In all cases the HTTP(S) status messages and error codes defined in **[RFC2616]** apply.

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

# 7 XRI Resolution Flow

970  
971  
972  
973

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*.



974

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

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

- 978 • Figure 3: Input processing (section 8.1).
- 979 • Figure 4: Output processing (section 8.2).
- 980 • **Figure 5: Authority resolution (section 9).**
- 981 • Figure 6: XRDS requests (section 9.1.3).
- 982 • **Figure 7: Redirect and Ref processing (section 12).**
- 983 • **Figure 8: Service endpoint selection (section 13).**
- 984 • Figure 9: Service endpoint selection logic (section 0).

985 **IMPORTANT:** In all cases the flowcharts are informative and the specification text is normative.  
986 However, the flowcharts are recommended as an aid in reading the specification. In particular,  
987 those highlighted in **bold** above illustrate the recursive calls for authority resolution and service  
988 endpoint selection used during Redirect and Ref processing (section 12). Implementers should  
989 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

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

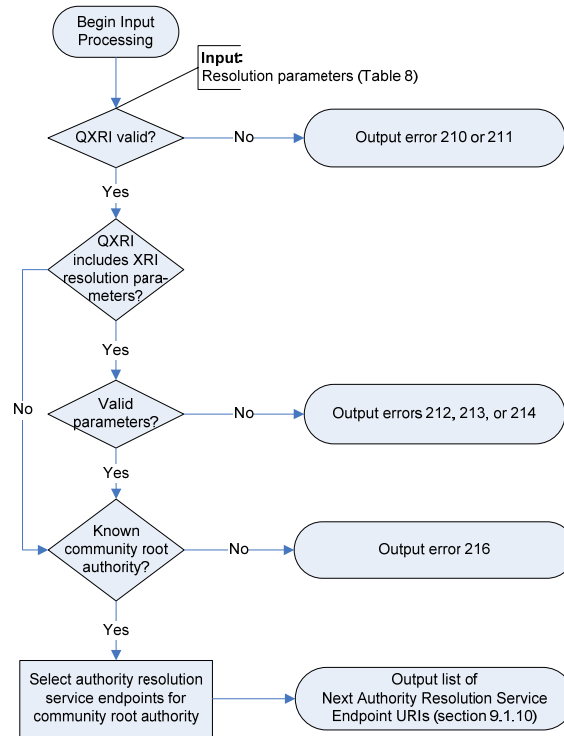
Logical Input Parameter Name	Type	Required/Optional	Default	Resolution Phase	Section
QXRI (query XRI) including Authority String, Path String, and Query String	xs:anyURI	Required	N/A	Authority Resolution (except Path String which is used in Service Endpoint Selection)	8.1.1
Resolution Output Format	xs:string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

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:

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

1014 Figure 3 is a flowchart (non-normative) illustrating the processing of input parameters.



1015

1016 *Figure 3: Input processing flowchart.*

1017 The following sections specify additional validation and usage requirements that apply to  
1018 particular input parameters.

1019 **8.1.1 QXRI (Authority String, Path String, and Query String)**

1020 The QXRI (query XRI) is the only REQUIRED input parameter. Per [XRISyntax], a QXRI consists  
 1021 of three logical subparameters as defined in Table 9.

Logical Parameter Name	Type	Required/Optional	Value
Authority String	xs:string	Required	Contents of the authority component of the QXRI, NOT including the XRI scheme name or leading double forward slashes ("/") or a terminating single forward slash ("/").
Path String	xs:string	Optional	Contents of the path component of the QXRI, NOT including the leading single forward slash ("/") or terminating delimiter (such as "/", "?", "#", whitespace, or CRLF). If the path component is absent or empty, the value is null.
Query String	xs:string	Optional	Contents of the query component of the QXRI, NOT including leading question mark ("?") or terminating delimiter (such as "#", white space, or CRLF). If the query component is absent or empty, the value is null.

1022 *Table 9: Subparameters of the QXRI input parameter.*

1023 The fourth possible component of a QXRI—a fragment—is by definition resolved locally relative  
 1024 to the target resource identified by the combination of the Authority, Path, and Query  
 1025 components, and as such does not play a role in XRI resolution.

1026 Following are the constraints on the value of the QXRI parameter.

- 1027 1. It MUST be a valid absolute XRI according to the ABNF defined in [XRISyntax]. To  
 1028 resolve a relative XRI reference, it must be converted into an absolute XRI using the  
 1029 procedure defined in section 2.4 of [XRISyntax].
- 1030 2. For authority or proxy resolution as defined in this specification, the QXRI MUST be in  
 1031 URI-normal form as defined in section 2.3.1 of [XRISyntax]. A local resolver API MAY  
 1032 support the input of other XRI forms but SHOULD document the normal form(s) it  
 1033 supports and its normalization policies.
- 1034 3. When a QXRI is included as part of an HXRI (section 11.2) for XRI proxy resolution, the  
 1035 QXRI MUST be normalized as specified in section 11.2, and all HXRI query parameters  
 1036 MUST follow the encoding rules specified in sections 11.3 and 11.4.

1037 **8.1.2 Resolution Output Format**

1038 The Resolution Output Format is an OPTIONAL parameter that, together with its subparameters,  
 1039 is used to specify:

- 1040 • The media type for the resolution response.
- 1041 • Whether generic or trusted resolution must be used by the resolver.
- 1042 • Whether Refs should be followed during resolution.
- 1043 • Whether CanonicalID verification should not be performed during resolution.
- 1044 • Whether service endpoint selection should be performed on the final XRD.

- 1045 • Whether default matches should be ignored during service endpoint selection.
- 1046 • Whether URIs should automatically be constructed in the final XRD.

1047 Following are the normative requirements for the use of this parameter.

- 1048 1. The Resolution Output Format MUST be one of the values specified in Table 5 and MAY  
1049 include any of the subparameters specified in Table 6.
- 1050 2. If the value of the `https` subparameter is TRUE, the resolver MUST use the HTTPS  
1051 trusted authority resolution protocol specified in section 10.1 (or return an error indicating  
1052 this is not supported).
- 1053 3. If the value of the `saml` subparameter is TRUE, the resolver MUST use the SAML trusted  
1054 authority resolution protocol specified in section 10.2 (or return an error indicating this is  
1055 not supported).
- 1056 4. If the value of both the `https` and `saml` subparameters are TRUE, the resolver MUST  
1057 use the HTTPS+SAML trusted authority resolution protocol specified in section 10.3 (or  
1058 return an error indicating this is not supported).
- 1059 5. If the value of the `cid` subparameter is TRUE or null, or if the parameter is absent, the  
1060 resolver MUST perform CanonicalID verification as specified in section 14.3. If the value  
1061 of the `cid` subparameter is FALSE, the resolver MUST NOT perform CanonicalID  
1062 verification.
- 1063 6. If the value of the `refs` subparameter is TRUE or null, or if the parameter is absent, the  
1064 resolver MUST perform Ref processing as specified in section 12. If the value of the  
1065 `refs` subparameter is FALSE, the resolver MUST NOT perform Ref processing and  
1066 must return an error if a Ref is encountered as specified in section 12.
- 1067 7. If the value of the `sep` subparameter is TRUE, the resolver MUST perform service  
1068 endpoint selection on the final XRD (even if the values of all service endpoint selection  
1069 parameters are null). If the value of the `sep` subparameter is FALSE or null, or if the  
1070 parameter is absent, the resolver MUST NOT perform service endpoint selection on the  
1071 final XRD unless it is required to produce a URI List or HTTP(S) redirect. See section 8.2.
- 1072 8. If the value of the `nodefault_t`, `nodefault_p`, or `nodefault_m` subparameter is  
1073 TRUE, the resolver MUST ignore default matches on the corresponding service endpoint  
1074 selection element categories as specified in section 13.2.2.
- 1075 9. If the value of the `uric` subparameter is TRUE, the resolver MUST perform service  
1076 endpoint URI construction as specified in section 13.6.1. If the value of the `uric`  
1077 subparameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT  
1078 perform service endpoint URI construction.

1079 Future versions of this specification, or other specifications for XRI resolution, MAY use other  
1080 values for Resolution Output Format or its subparameters.

### 1081 8.1.3 Service Type

1082 The Service Type is an OPTIONAL value of type `xs:anyURI` used to request a specific type of  
1083 service in the service endpoint selection phase (section 11). The value of this parameter MUST  
1084 be a valid absolute XRI, IRI, or URI in URI-normal form as defined by **[XRISyntax]**. (Note that  
1085 URI-normal form is required so this parameter may be passed to a proxy resolver in a QXRI  
1086 query parameter as defined in section 11.) The Service Type values defined for XRI resolution  
1087 services are specified in section 3.1.2. The rules for matching the value of the Service Type  
1088 parameter to the value of the `xrd:XRD/xrd:Service/xrd:Type` element are specified in  
1089 section 13.2.6.

#### 1090 **8.1.4 Service Media Type**

1091 The Service Media Type is an OPTIONAL string used to request a specific media type in the  
1092 service endpoint selection phase (section 11). The value of this parameter MUST be a valid  
1093 media type as defined by **[RFC2046]**. The Service Media Type values defined for XRI resolution  
1094 services are specified in section 3.3. The rules for matching the value of the Service Media Type  
1095 parameter to the value of the `xrd:XRD/xrd:Service/xrd:MediaType` element are specified  
1096 in section 13.2.8.

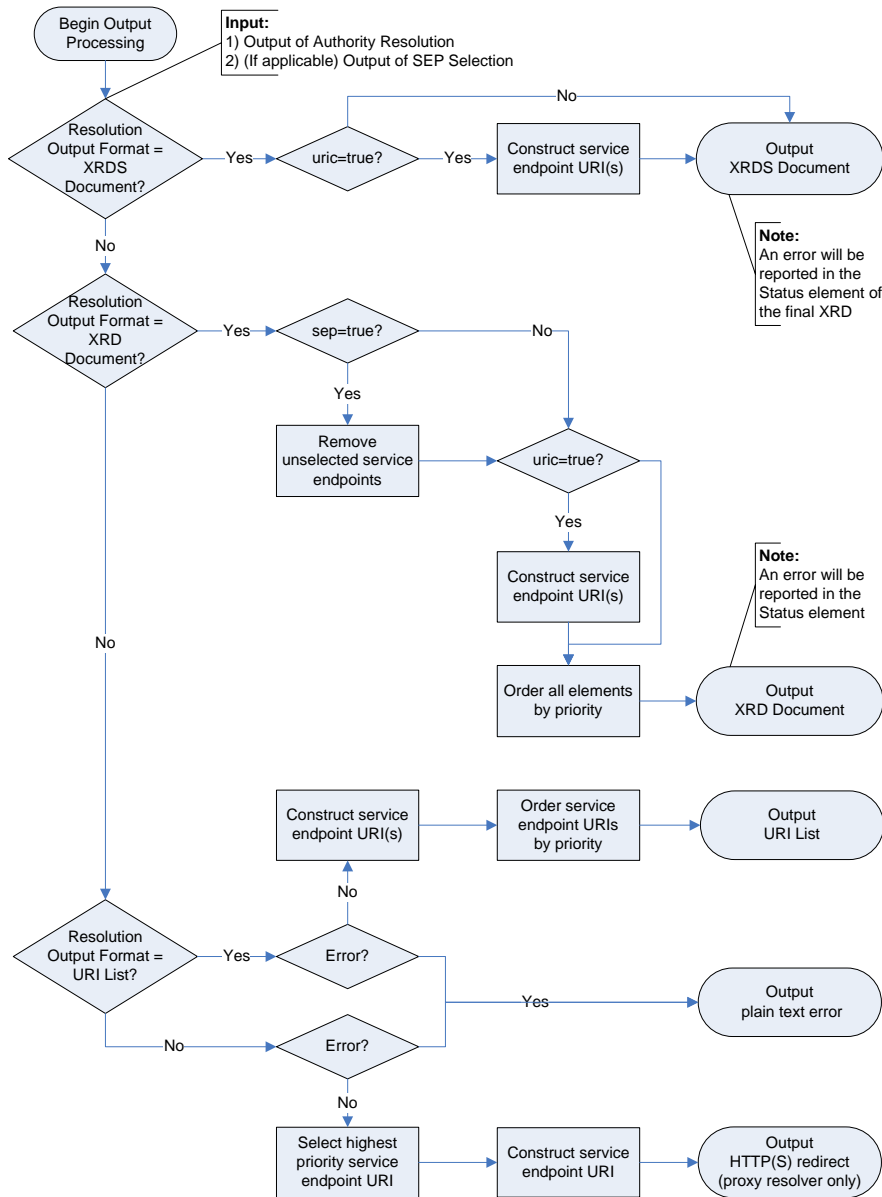
1097 **8.2 Outputs**

1098 Table 10 summarizes the logical outputs of XRI resolution. Note that these are defined in terms of  
1099 media types returned by authority servers and proxy resolvers. A local resolver API MAY  
1100 implement other representations of these media types.

Logical Output Format Name	Media Type Value (when requesting XRI authority resolution only)	Media Type Value (when requesting service endpoint selection)
XRDS Document	application/xrds+xml	application/xrds+xml;sep=true
XRD Element	application/xrd+xml	application/xrd+xml;sep=true
URI List	N/A	text/uri-list
HTTP(S) Redirect	N/A	<i>null</i>

1101 Table 10: Outputs of XRI resolution.

1102 Figure 4 is a flowchart illustrating the process of producing these output formats once the auth-  
 1103 ority resolution and optional service endpoint selection phases are complete. Note that in the first  
 1104 two output options, errors are reported directly in the XRDS, so no special error format is needed.



1105  
 1106 *Figure 4: Output processing flowchart.*

1107 The following sections provide additional construction and validation requirements.

## 1108 8.2.1 XRDS Document

1109 If the value of the Resolution Output Format parameter is `application/xrds+xml`, the  
1110 following rules apply.

- 1111 1. The output MUST be a valid XRDS document according to the schema defined in  
1112 Appendix B.
- 1113 2. The XRDS document MUST contain an ordered list of `xrd:XRD` elements—one for each  
1114 authority subsegment successfully resolved by the resolver client. This list MUST appear  
1115 in the same order as the corresponding subsegments in the Authority String.
- 1116 3. Each of the contained XRD elements must be a valid XRD element according to the  
1117 schema defined in Appendix B.
- 1118 4. The XRD elements MUST conform to the additional requirements in section 4.
- 1119 5. If the value of the `saml` subparameter of the Resolution Output Format is TRUE, the  
1120 XRD elements MUST conform to the additional requirements in section 10.2.
- 1121 6. If Redirect or Ref processing is necessary during the authority resolution or service  
1122 endpoint selection process, it MUST result in a valid nested XRDS document as defined  
1123 in section 12.
- 1124 7. If the value of the `sep` subparameter is TRUE, service endpoint selection MUST be  
1125 performed as defined in section 13, even if the values of all three service endpoint  
1126 selection input parameters (Service Type, Path String, and Service Media Type) are null.

1127 **IMPORTANT:** No filtering of the final XRD is performed when returning an XRDS document.  
1128 Filtering is only performed when the requested Resolution Output Format is an XRD element –  
1129 see the next section.

- 1130 8. If the value of the `cid` subparameter is TRUE, synonym verification MUST be reported  
1131 using the `xrd:Status` element of each XRD in the XRDS document as defined in  
1132 section 14.
- 1133 9. If the output is an error, this error MUST be returned using the `xrd:Status` element of  
1134 the final XRD in the XRDS document as defined in section 15.

## 1135 8.2.2 XRD Element

1136 If the value of the Resolution Output Format parameter is `application/xrd+xml`, the following  
1137 rules apply.

- 1138 1. The output MUST be a valid XRD element according to the schema defined in Appendix  
1139 B.
- 1140 2. The XRD elements MUST conform to the additional requirements in section 4.
- 1141 3. If the value of the `saml` subparameter of the Resolution Output Format is TRUE, the  
1142 XRD element MUST conform to the additional requirements in section 10.2.
- 1143 4. If the value of the `sep` subparameter is FALSE or null, or if this parameter is absent, the  
1144 XRD MUST be the final XRD in the XRDS document produced as a result of authority  
1145 resolution. Service endpoint selection or any other filtering of the XRD element MUST  
1146 NOT be performed.
- 1147 5. If the value of the `sep` subparameter is TRUE, service endpoint selection MUST be  
1148 performed as defined in section 13, even if the values of all service endpoint selection  
1149 input parameters are null.
- 1150 6. If service endpoint selection is performed, the only `xrd:Service` elements in the XRD  
1151 element MUST be those selected according to the rules specified in section 13. If no  
1152 service endpoints were selected by those rules, no `xrd:Service` elements will be

1153 present. In addition, all elements within the XRD element that are subject to the global  
1154 `priority` attribute (even if the attribute is absent or null) MUST be returned in order of  
1155 highest to lowest priority as defined in section 4.3.3.

1156 **IMPORTANT:** Any other filtering of the XRD element MUST NOT be performed. Note that this  
1157 means that if the XRD element includes a SAML signature element as defined in section 10.2,  
1158 this element is still returned inside the XRD element even though it may not be able to be verified  
1159 by a consuming application.

- 1160 7. If the value of the `cid` subparameter is TRUE, synonym verification MUST be reported  
1161 using the `xrd:Status` element of each XRD in the XRDS document as defined in  
1162 section 14.
- 1163 8. If the output is an error, this error MUST be returned using the `xrd:Status` element as  
1164 defined in section 15.

### 1165 8.2.3 URI List

1166 If the value of the Resolution Output Format parameter is `text/uri-list`, the following rules  
1167 apply.

- 1168 1. For this output, service endpoint selection is REQUIRED, even if the values of all service  
1169 endpoint selection input parameters are null.
- 1170 2. If authority resolution and service endpoint selection are both successful, the output  
1171 MUST be a valid URI List as defined by section 5 of [RFC2483].
- 1172 3. If, after applying the service endpoint selection rules, more than one service endpoint is  
1173 selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as  
1174 defined in section 4.3.3.
- 1175 4. If the final selected `xrd:XRD/xrd:Service` element contains a  
1176 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`  
1177 element, Redirect and Ref processing MUST be performed as described in section 12.  
1178 This rule applies iteratively to each new XRDS document resolved.
- 1179 5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)  
1180 MUST be constructed as defined in section 13.6.1.
- 1181 6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`  
1182 elements within the selected `xrd:Service` element as defined in section 4.3.3. When  
1183 two or more of the source `xrd:URI` elements have equal priority, their constructed URIs  
1184 SHOULD be returned in random order.

1185 **IMPORTANT:** Any other filtering of the URI list MUST NOT be performed.

- 1186 7. If the output is an error, it MUST be returned with the content type `text/plain` as  
1187 defined in section 15.

### 1188 8.2.4 HTTP(S) Redirect

1189 In XRI proxy resolution, the Resolution Output Format parameter may be null. In this case the  
1190 output of a proxy resolver is an HTTP(S) redirect as defined in section 11.7.

1191

## 9 Generic Authority Resolution Service

1192  
1193  
1194  
1195

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

1196  
1197  
1198  
1199  
1200

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

1201

### 9.1 XRI Authority Resolution

1202

#### 9.1.1 Service Type and Service Media Type

1203

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)

1204

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

1205  
1206

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

1207  
1208  
1209  
1210  
1211

**BACKWARDS COMPATABILITY 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:

1212  
1213  
1214  
1215  
1216  
1217

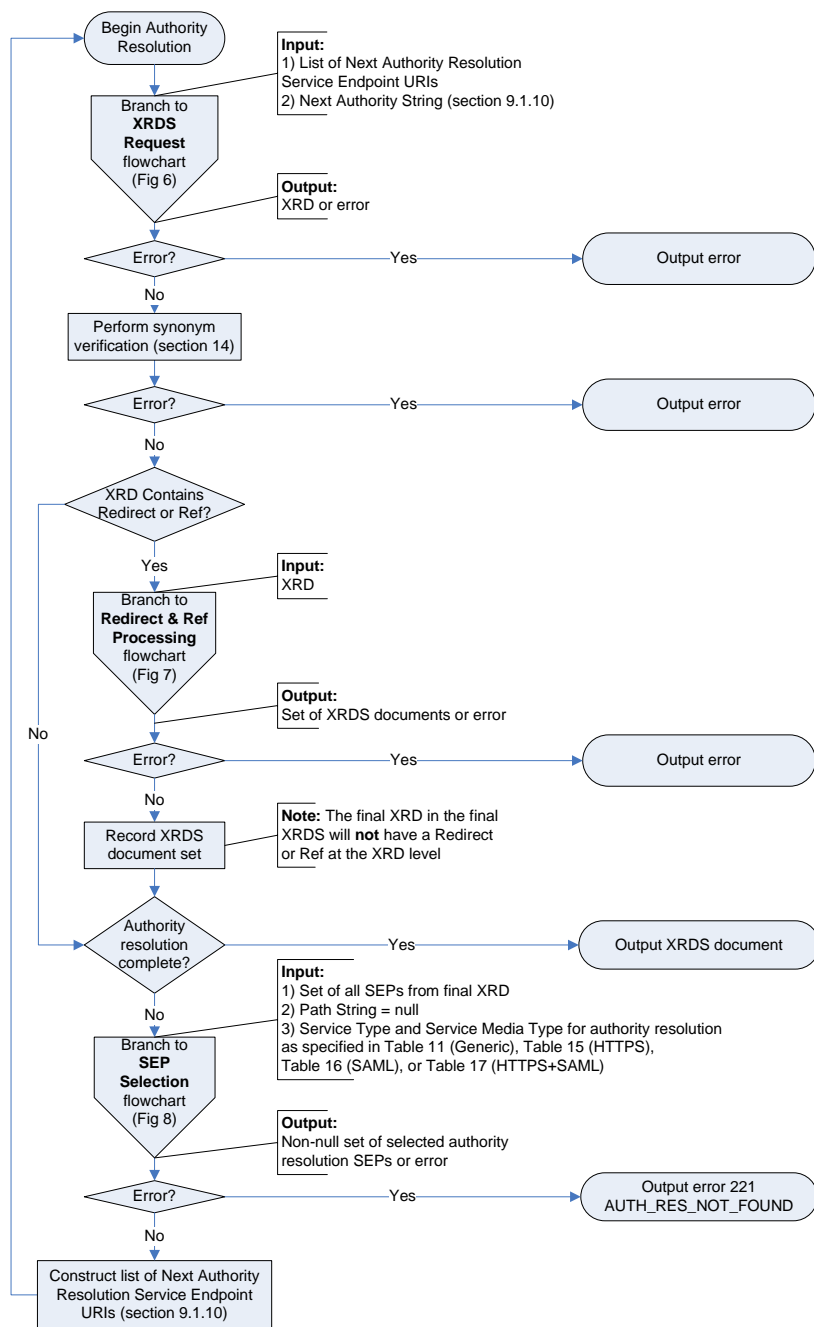
```

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

```

1218 **9.1.2 Protocol**

1219 Figure 5 (non-normative) illustrates the overall logical flow of generic authority resolution.



1220

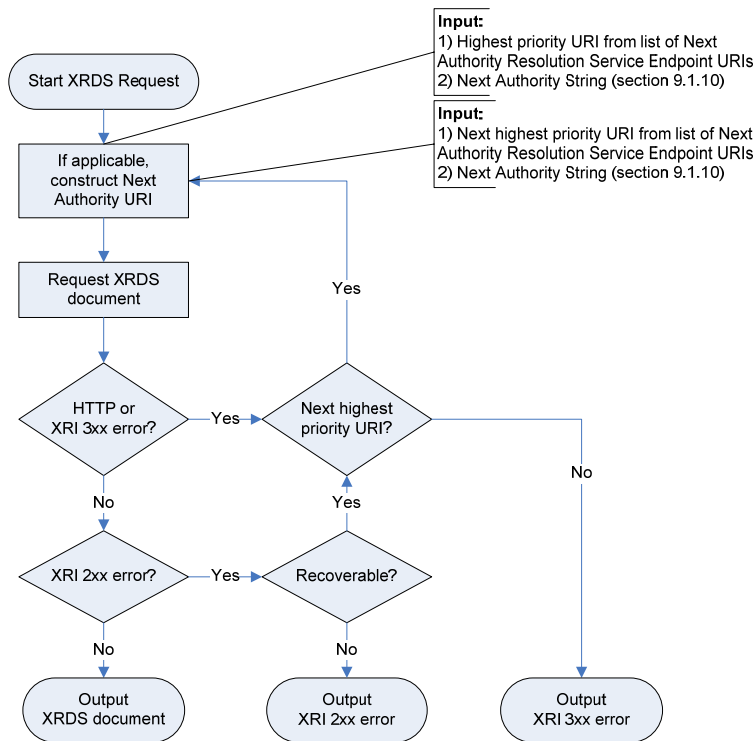
1221 *Figure 5: Authority resolution flowchart.*

1222 Following are the normative requirements for behavior of an XRI resolver and an XRI authority  
1223 server when performing generic XRI authority resolution:

- 1224 1. Each request for an XRDS document using HTTP(S) MUST conform to the requirements  
1225 in section 9.1.3.
- 1226 2. For errors in XRDS document resolution requests, a resolver MUST implement failover  
1227 handling as specified in section 9.1.4.
- 1228 3. The resolver MUST be preconfigured with or have a means of obtaining the XRDS  
1229 document describing the community root authority for the XRI to be resolved as defined  
1230 in section 9.1.5.
- 1231 4. The resolver MAY obtain the XRDS document describing the community root authority by  
1232 requesting a self-describing XRDS document as defined in section 9.1.6.
- 1233 5. Resolution of each subsegment in the Authority String after the community root  
1234 subsegment MUST proceed in subsegment order (left-to-right) using fully qualified  
1235 subsegment values as defined in section 9.1.7.
- 1236 6. Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as  
1237 defined in section 9.1.8.
- 1238 7. For each iteration of the authority resolution process, the next authority resolution service  
1239 endpoint MUST be selected as specified in section 9.1.9.
- 1240 8. For each iteration of the authority resolution process, an HTTP(S) URI called the Next  
1241 Authority URI MUST be constructed according to the algorithm specified in section  
1242 9.1.10.
- 1243 9. A resolver MAY request that a recursing authority server perform resolution of multiple  
1244 subsegments as defined in section 9.1.11.
- 1245 10. For each iteration of the authority resolution process, a resolver MUST perform Redirect  
1246 and Ref processing as specified in section 12. Note that if Redirect and Ref processing is  
1247 successful, it will result in a nested XRDS document as specified in section 12.5.

### 1248 **9.1.3 Requesting an XRDS Document using HTTP(S)**

1249 Figure 6 (non-normative) illustrates the logical flow for requesting an XRDS document.



1250  
1251

Figure 6: XRDS request flowchart.

1252  
1253

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

1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274

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.

- 1275 5. If all authority resolution service endpoints fail, the resolver SHOULD return the  
1276 appropriate error code and context message as specified in section 15. In recursing  
1277 resolution, such an error MUST be returned by the recursing authority server to the  
1278 resolver as specified in section 15.4.
- 1279 6. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section  
1280 16. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent  
1281 possible to maintain the efficiency and scalability of the HTTP-based resolution system.  
1282 The recommended use of HTTP caching headers is described in more detail in section  
1283 16.2.1.

#### 1284 9.1.4 Failover Handling

1285 XRI infrastructure has the same requirements as DNS infrastructure for stability, redundancy, and  
1286 network performance. This means XRI authority and proxy resolution services are subject to the  
1287 same requirements as DNS nameservers. For example:

- 1288 • Critical authority or proxy resolution servers SHOULD be operated from a minimum of two  
1289 physically separate network locations to prevent a single point of failure.
- 1290 • Authority or proxy resolution servers handling heavy loads SHOULD operate from multiple  
1291 servers and take advantage of load balancing technologies.

1292 However, such capabilities are effective only if resolvers or other client applications implement  
1293 proper failover handling. Because XRI resolution takes place at a layer above DNS resolution,  
1294 resolvers have two ways to discover additional network endpoints at which authority or proxy  
1295 resolution services are available.

- 1296 • *DNS round robin/failover*: The domain name of an authority resolution service endpoint URI  
1297 may be associated with more than one IP address.
- 1298 • *XRI round robin/failover*: The XRDS document describing an XRI authority may publish  
1299 multiple URI elements for its authority resolution service endpoint, or multiple authority  
1300 resolution service endpoints, or both.

1301 To take advantage of both these options, the following rules apply to failover handling:

- 1302 1. A resolver SHOULD first try an alternate IP address for the current authority resolution  
1303 service endpoint if the endpoint uses DNS round robin.
- 1304 2. If all alternate IP addresses fail, a resolver MUST try the next highest priority authority  
1305 resolution URI in the current authority resolution service endpoint, if available.
- 1306 3. If all URIs in the current authority resolution service endpoint fail, a resolver MUST try the  
1307 next highest priority authority resolution service endpoint, if available, until all authority  
1308 resolution service endpoints are exhausted.
- 1309 4. A resolver SHOULD only return an error if all network endpoints associated with the  
1310 authority resolution service fail to respond.

1311 **IMPORTANT:** These rules also apply to any client of an XRI proxy resolver. Failure to observe  
1312 this warning means the proxy resolver can become a point of failure.

1313 One final consideration: DNS caching mechanisms should respect the TTL (Time To Live)  
1314 settings in DNS records. However, different software languages and frameworks handle DNS  
1315 caching differently. It is RECOMMENDED to check the default settings to ensure that a library or  
1316 application is not caching DNS results indefinitely.

#### 1317 9.1.5 Community Root Authorities

1318 Identifier management policies are defined on a community-by-community basis. For XRI  
1319 identifier authorities, the resolution community is specified by the first (leftmost) subsegment of

1320 the authority component of the XRI. This is referred to as the *community root authority*, and it  
1321 represents the authority server(s) that answer resolution queries at this root. When a resolution  
1322 community chooses to create a new community root authority, it SHOULD define policies for  
1323 assigning and managing identifiers under this authority. Furthermore, it SHOULD define what  
1324 resolution protocol(s) may be used for these identifiers.

1325 For an XRI authority, the community root may be either a global context symbol (GCS) character  
1326 or top-level cross-reference as specified in section 2.2.1.1 of [XRISyntax]. In either case, the  
1327 corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution  
1328 service endpoints for that community.

1329 The community root authority SHOULD publish a self-describing XRDS document as defined in  
1330 section 9.1.6. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as  
1331 the community's root authority resolution service endpoints. This community root XRDS  
1332 document, or its location, must be known *a priori* and is part of the configuration of an XRI  
1333 resolver, similar to the specification of root DNS servers for a DNS resolver. Note that it is not  
1334 strictly necessary to publish this information in an XRDS document—it may be supplied in any  
1335 format that enables configuration of the XRI resolvers in the community. However, publishing a  
1336 self-describing XRDS document at a known location simplifies this process and enables dynamic  
1337 configuration of community resolvers.

1338 As a best practice, it is RECOMMENDED that community root XRDS document contain:

- 1339 • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.
- 1340 • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML  
1341 trusted resolution is supported.
- 1342 • Both of the above if HTTPS+SAML trusted resolution is supported.
- 1343 • The service endpoints and supported media types of the community's XRI proxy resolver(s) if  
1344 proxy resolution is supported.

1345 For a list of public community root authorities and the locations of their community root XRDS  
1346 documents, see the Wikipedia entry on XRI [WikipediaXRI].

### 1347 **9.1.6 Self-Describing XRDS Documents**

1348 An identifier authority MAY publish a self-describing XRDS document, i.e., one produced by the  
1349 same identifier authority that it describes. A resolver MAY request a self-describing XRDS  
1350 document from a target identifier authority using either of two methods:

- 1351 1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution  
1352 service endpoint, it may use the resolution protocol specified in section 6 to request an  
1353 XRDS document directly from this HTTP(S) URI. This HTTP(S) URI may be known a  
1354 priori (as is often the case with community root authorities, above), or it may be  
1355 discovered from other identifier authorities via the resolution protocols defined in this  
1356 specification.
- 1357 2. If the resolver knows: a) an XRI of the target authority as a community root authority, and  
1358 b) an HTTP(S) URI for a proxy resolver configured for this community root authority, it  
1359 may use the proxy resolution protocol specified in section 11 to query the proxy resolver  
1360 for the community root authority XRI. This query MUST include only a single subsegment  
1361 identifying the community root authority and MUST NOT include any additional  
1362 subsegments.

1363 If an identifier authority had an authority resolution service endpoint at  
1364 `http://example.com/auth-res-service/`, an example of the first method would be to  
1365 issue an HTTP(S) GET request to that URI with an Accept header specifying the content type  
1366 `application/xrds+xml`. See section 6.3 for more details.

1367 If an identifier authority with the community root authority identifier `xri://(example)` was  
1368 registered with the XRI proxy resolver `http://xri.example.com/`, an example of the second  
1369 method would be to issue an HTTP(S) GET request to the following URI:

1370 `http://xri.example.com/(example)?_xrd_r=application/xrds+xml`

1371 Note that a proxy resolver may use the first method to publish its own self-describing XRDS  
1372 document at the HTTP(S) URI(s) for its proxy resolution service.

1373 **IMPORTANT:** A self-describing XRDS document **MUST** only be issued by an identifier authority  
1374 when describing itself. It **MUST NOT** be included in an XRDS document when describing a  
1375 different identifier authority. In the latter case the self-describing XRDS document for the  
1376 community root authority is implicit.

### 1377 **9.1.7 Qualified Subsegments**

1378 A qualified subsegment is defined by the productions whose names start with `xri-subseg` in  
1379 section 2.2.3 of **[XRISyntax]** *including the leading syntactic delimiter* (“\*” or “!”). A qualified  
1380 subsegment **MUST** include the leading syntactic delimiter even if it was optionally omitted in the  
1381 original XRI (see section 2.2.3 of **[XRISyntax]**).

1382 If the first subsegment of an XRI authority is a GCS character and the following subsegment does  
1383 not begin with a “\*” (indicating a reassignable subsegment) or a “!” (indicating a persistent  
1384 subsegment), then a “\*” is implied and **MUST** be added when constructing the qualified  
1385 subsegment as specified in section 9.1.7. Table 12 and Table 13 illustrate the differences  
1386 between parsing a reassignable subsegment following a GCS character and parsing a cross-  
1387 reference, respectively.

1388

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

1389 *Table 12: Parsing the first subsegment of an XRI that begins with a global context symbol.*

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

1390 *Table 13: Parsing the first subsegment of an XRI that begins with a cross-reference.*1391 

### 9.1.8 Cross-References

1392 Any subsegment within an XRI authority component may be a cross-reference (see section 2.2.2  
 1393 of [XRISyntax]). Cross-references are resolved identically to any other subsegment because the  
 1394 cross-reference is considered opaque, i.e., the value of the cross-reference (including the  
 1395 parentheses) is the literal value of the subsegment for the purpose of resolution.

1396 Table 14 provides several examples of resolving cross-references. In these examples,  
 1397 subsegment !b resolves to a Next Authority Resolution Service Endpoint URI of  
 1398 `http://example.com/xri/` and recursing authority resolution is not being requested.  
 1399

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

1400 *Table 14: Examples of the Next Authority URIs constructed using different types of cross-references.*1401 

### 9.1.9 Selection of the Next Authority Resolution Service Endpoint

1402 For each iteration of authority resolution, the resolver MUST select the next authority resolution  
 1403 service endpoint from the current XRD as specified in section 13. For generic authority resolution,  
 1404 this selection process MUST use the parameters specified in Table 11. For trusted authority  
 1405 resolution, this selection process MUST use the parameters specified in Table 15, Table 16, or  
 1406 Table 17. In all cases, an explicit match on the `xrd:XRD/xrd:Service/xrd:Type` element is  
 1407 REQUIRED, so during authority resolution, a resolver MUST set the `nodefault` parameter to a  
 1408 value of `nodefault=type` in order to override selection of a default service endpoint as  
 1409 specified in section 13.2.2.

## 1410 9.1.10 Construction of the Next Authority URI

1411 Once the next authority resolution service endpoint is selected, the resolver MUST construct a  
1412 URI for the next HTTP(S) request, called the *Next Authority URI*, by concatenating two strings as  
1413 specified in this section.

1414 The first string is called the *Next Authority Resolution Service Endpoint URI*. To construct it, the  
1415 resolver MUST:

- 1416 1. Select the highest priority URI of the highest priority authority resolution service endpoint  
1417 selected in section 9.1.9.
- 1418 2. Apply the service endpoint URI construction algorithm based the value of the `append`  
1419 attribute as defined in section 13.6.
- 1420 3. Append a forward slash ("/") *if the URI does not already end in a forward slash*.

1421 The second string is called the *Next Authority String* and it consists of either:

- 1422 • The next fully qualified subsegment to be resolved (see section 9.1.7), or
- 1423 • In the case of recursing resolution, the next fully qualified subsegment to be resolved plus  
1424 any additional subsegments for which recursing resolution is requested (see section 9.1.11).

1425 The final step is to append the Next Authority String to the path component of the Next Authority  
1426 Resolution Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

1427 Construction of the Next Authority URI is more formally described in this pseudocode for  
1428 resolving a "next-auth-string" via a "next-auth-res-sep-uri":

```
1429 if (path portion of next-auth-res-sep-uri does not end in "/"):
1430     append "/" to path portion of next-auth-res-sep-uri
1431
1432 if (next-auth-string is not preceded with "*" or "!" delimiter):
1433     prepend "*" to next-auth-string
1434
1435 append uri-escape(next-auth-string) to path of next-auth-res-sep-uri
```

## 1436 9.1.11 Recursing Authority Resolution

1437 If an authority server offers recursing resolution, an XRI resolver MAY request resolution of  
1438 multiple authority subsegments in one transaction. If a resolver makes such a request, the  
1439 responding authority server MAY perform the additional recursing resolution steps requested. In  
1440 this case the recursing authority server acts as a resolver to the other authority resolution service  
1441 endpoints that need to be queried. Alternatively, the recursing authority server may retrieve XRDS  
1442 from its local cache until it reaches a subsegment whose XRD is not locally cached, or it may  
1443 simply recurse only as far as it is authoritative.

1444 If an authority server performs any recursing resolution, it MUST return an ordered list of  
1445 `xrd:XRDS` elements (and nested `xrd:XRDS` elements if Redirects or Refs are followed as  
1446 specified in section 12) in an `xrd:XRDS` document for all subsegments resolved as defined in  
1447 section 8.2.1.

1448 A recursing authority server MAY resolve fewer subsegments than requested by the resolver. The  
1449 recursing authority server is under no obligation to resolve more than the first subsegment (for  
1450 which it is, by definition, authoritative).

1451 If the recursing authority server does not resolve the entire set of subsegments requested, the  
1452 resolver MUST continue the authority resolution process itself. At any stage, however, the  
1453 resolver MAY request recursing resolution of any or all of the remaining authority subsegments.

## 1454 9.2 IRI Authority Resolution

1455 From the standpoint of generic authority resolution, an IRI authority component represents either  
1456 a DNS name or an IP address at which an XRDS document describing the authority may be  
1457 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET  
1458 request to a URI constructed from the IRI authority component. The resulting XRDS document  
1459 can then be consumed in the same manner as one obtained using XRI authority resolution.

1460 While the use of IRI authorities provides backwards compatibility with the large installed base of  
1461 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of  
1462 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities  
1463 are NOT RECOMMENDED for new deployments of XRI identifiers.

1464 This section defines IRI authority resolution as a simple extension to the XRI authority resolution  
1465 protocol defined in the preceding section.

### 1466 9.2.1 Service Type and Media Type

1467 Because IRI authority resolution takes place at a level “below” XRI authority resolution, it cannot  
1468 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority  
1469 resolution uses the same media type as generic XRI authority resolution.

### 1470 9.2.2 Protocol

1471 Following are the normative requirements for IRI authority resolution that differ from generic XRI  
1472 authority resolution:

1473 1. The Next Authority URI (section 9.1.10) is constructed by extracting the entire IRI  
1474 authority component and prepending the string `http://`. See the exception in section  
1475 9.2.3.

1476 2. The HTTP GET request MUST include an HTTP Accept header containing only the  
1477 following:

```
1478 Accept: application/xrds+xml
```

1479 3. The HTTP GET request MUST have a `Host:` header (as defined in section 14.23 of  
1480 [RFC2616]) containing the value of the IRI authority component. For example:

```
1481 Host: example.com
```

1482 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document  
1483 containing the XRD describing that authority.

1484 5. The responding server MUST use the value of the `Host:` header to populate the  
1485 `xrd:XRD/xrd:Query` element in the resulting XRD.

1486 Note that because IRI authority resolution is required to process the entire IRI authority  
1487 component in a single step, recursing authority resolution does not apply.

### 1488 9.2.3 Optional Use of HTTPS

1489 Section 10 of this specification defines trusted resolution only for XRI authorities. Trusted  
1490 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to  
1491 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY  
1492 use HTTPS as the access protocol for retrieving the authority's XRD. If the resolver is satisfied,  
1493 via transport level security mechanisms, that the response is from the expected IRI authority, the  
1494 resolver MAY consider this an HTTPS trusted resolution response as defined in section 10.1.

1495

## 10 Trusted Authority Resolution Service

1496  
1497  
1498

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

1499

### 10.1 HTTPS

1500  
1501  
1502  
1503

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

1504

#### 10.1.1 Service Type and Service Media Type

1505

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

1506

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

1507  
1508  
1509  
1510

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

1511

#### 10.1.2 Protocol

1512  
1513

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

1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525

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.

1526

### 10.2 SAML

1527  
1528  
1529  
1530  
1531

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 is RECOMMENDED to combine SAML trusted resolution with

1532 HTTPS trusted resolution as defined in section 10.3. Message confidentiality may also be  
1533 achieved with other security protocols used in conjunction with this specification. SAML trusted  
1534 resolution also does not provide a means for an authority to provide different responses for  
1535 different requestors; client authentication is explicitly out-of-scope for version 2.0 of XRI  
1536 resolution.

## 1537 10.2.1 Service Type and Service Media Type

1538 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

1539 *Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.*

1540 A SAML trusted resolution service endpoint advertised in an XRDS document MUST use the  
1541 Service Type identifier and Service Media Type identifier defined in Table 16 (including the  
1542 `saml=true` subparameter). In addition, for transport security the identifier authority SHOULD  
1543 offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

## 1544 10.2.2 Protocol

### 1545 10.2.2.1 Client Requirements

1546 For a resolver, trusted resolution is identical to the generic resolution protocol (section 9.1) with  
1547 the addition of the following requirements:

- 1548 1. All authority resolution service endpoints MUST be selected using the values defined in  
1549 Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an  
1550 authority unless the authority advertises a resolution service endpoint matching these  
1551 values.
- 1552 2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is  
1553 RECOMMENDED for confidentiality.
- 1554 3. All authority resolution requests MUST contain an HTTP(S) Accept header with the  
1555 media type identifier defined in Table 16 (including the `saml=true` subparameter). This  
1556 is the media type of the requested response.

1557 **IMPORTANT:** Clients willing to accept either generic or trusted responses MAY use a  
1558 combination of media type identifiers in the Accept header as described in section 14.1 of  
1559 [RFC2616]. Media type identifiers SHOULD be ordered according to the client's preference for  
1560 the media type of the response. If a client performing generic authority resolution receives an  
1561 XRD containing SAML elements, it MAY choose not to validate the signature or perform any  
1562 processing of these elements.

- 1563 4. A resolver MAY request recursing authority resolution of multiple subsegments as  
1564 defined in section 10.2.3.
- 1565 5. The resolver MUST individually validate each XRD it receives in the resolution chain  
1566 according to the rules defined in section 10.2.4. When `xrd:XRD` elements come both  
1567 from freshly-retrieved XRDS documents and from a local cache, a resolver MUST ensure  
1568 that these requirements are satisfied each time a resolution request is performed.

### 1569 10.2.2.2 Server Requirements

1570 For an authority server, trusted resolution is identical to the generic resolution protocol (section  
1571 9.1) with the addition of the following requirements:

- 1572 1. The HTTP(S) response to a trusted resolution request MUST include a content type of  
1573 `application/xrds+xml;saml=true`.
- 1574 2. The XRDS document returned by the resolution service MUST contain a  
1575 `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid  
1576 per the processing rules described by [SAML].
- 1577 3. The `saml:Assertion` element MUST contain a valid enveloped digital signature as  
1578 defined by [XMLDSig] and as constrained by section 5.4 of [SAML].
- 1579 4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML  
1580 assertion. Specifically, the signature MUST contain a single  
1581 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference  
1582 MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML  
1583 assertion. The `URI` reference MUST NOT be empty and it MUST refer to the identifier  
1584 contained in the `xrd:XRD/@xml:id` attribute.
- 1585 5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain  
1586 a `ds:KeyInfo` element. If this element is included, it MUST describe the key used to  
1587 verify the digital signature element. However, because the signing key is known in  
1588 advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the  
1589 `ds:Signature` element of the SAML assertion.
- 1590 6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST  
1591 match the XRI authority subsegment requested by the client.
- 1592 7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match  
1593 the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD  
1594 advertising availability of trusted resolution service from this authority as required in  
1595 section 10.2.5.
- 1596 8. The `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be  
1597 present and equal to the `xrd:XRD/xrd:Query` element.
- 1598 9. The `NameQualifier` attribute of the  
1599 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be  
1600 present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.
- 1601 10. There MUST be exactly one `saml:AttributeStatement` present in the  
1602 `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute`  
1603 element with a `Name` attribute value of `xri://$xrd*($v*2.0)`. This  
1604 `saml:Attribute` element MUST contain exactly one `saml:AttributeValue`  
1605 element whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD`  
1606 element that is the immediate parent of the `saml:Assertion` element.

### 1607 10.2.3 Recursing Authority Resolution

1608 If a resolver requests trusted resolution of multiple authority subsegments (see section 9.1.8), a  
1609 recursing authority server SHOULD attempt to perform trusted resolution on behalf of the resolver  
1610 as described in this section. However, if the resolution service is not able to obtain trusted XRDs  
1611 for one or more additional recursing subsegments, it SHOULD return only the trusted XRDs it has  
1612 obtained and allow the resolver to continue.

## 1613 10.2.4 Client Validation of XRDs

1614 For each XRD returned as part of a trusted resolution request, the resolver MUST validate the  
1615 XRD according to the rules defined in this section.

- 1616 1. The `xrd:XRD/saml:Assertion` element MUST be present.
- 1617 2. This assertion MUST be valid per the processing rules described by [SAML].
- 1618 3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by  
1619 [XMLDSig] and constrained by Section 5.4 of [SAML].
- 1620 4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML  
1621 assertion. Specifically, the signature MUST contain a single  
1622 `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference  
1623 MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent  
1624 of the signed SAML assertion.
- 1625 5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo`  
1626 element, the resolver MAY reject the signature if this key does not match the signer's  
1627 expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor  
1628 that was used to describe the current authority. See section 10.2.5.
- 1629 6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose  
1630 resolution resulted in the current XRD.
- 1631 7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the  
1632 `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability  
1633 of trusted resolution service from this authority as required in section 10.2.5.
- 1634 8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the  
1635 `NameQualifier` attribute of the  
1636 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1637 9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the  
1638 `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.
- 1639 10. There MUST exist exactly one  
1640 `xrd:XRD/saml:Assertion/saml:AttributeStatment` with exactly one  
1641 `saml:Attribute` element that has a `Name` attribute value of `xri://$xrd*($v*2.0)`.  
1642 This `saml:Attribute` element must have exactly one `saml:AttributeValue`  
1643 element whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD`  
1644 element that is the immediate parent of the signed SAML assertion.

1645 If any of the above requirements are not met for an XRD in the trusted resolution chain, the result  
1646 MUST NOT be considered a valid trusted resolution response as defined by this specification.  
1647 Note that this does not preclude a resolver from considering alternative resolution paths. For  
1648 example, if an XRD advertising SAML trusted resolution service has two or more  
1649 `xrd:XRD/xrd:Service/xrd:URI` elements and the response from one service endpoint fails  
1650 to meet the requirements above, the client MAY repeat the validation process using the second  
1651 URI. If the second URI passes the tests, it MUST be considered a trusted resolution response as  
1652 defined by this document and SAML trusted resolution may continue.

1653 If the above requirements are met, and the `code` attribute of the `xrd:XRD/xrd:ServerStatus`  
1654 element is 100 (SUCCESS), the resolver MUST add an `xrd:XRD/xrd:Status` element  
1655 reporting a status of 100 (SUCCESS) as specified in section 15. Note that this added element  
1656 MUST be disregarded if a consuming application wishes to verify the SAML signature itself. (If  
1657 necessary, the consuming application may request the XRDS document it wishes to verify directly  
1658 from the SAML authority resolution server.)

1659 If all SAML trusted resolution paths fail, the resolver MUST return the appropriate 23x trusted  
1660 resolution error as defined in section 15.

1661 **10.2.5 Correlation of ProviderID and KeyInfo Elements**

1662 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at  
1663 least one unique persistent identifier expressed in the

1664 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority  
1665 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI  
1666 authority. While a ProviderID may be any valid URI that meets these requirements, it is  
1667 **STRONGLY RECOMMENDED** to use a persistent identifier such as a persistent XRI  
1668 **[XRI Syntax]** or a URN **[RFC2141]**.

1669 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in  
1670 an XRD advertising SAML trusted authority resolution service with the response received from a  
1671 SAML trusted resolution service endpoint. If the signed XRD response contains the same  
1672 ProviderID as the XRD used to advertise a service, and the resolver has reason to trust the  
1673 signature, the resolver can trust that the XRD response has not been maliciously replaced with  
1674 another XRD.

1675 There is no defined discovery process for the ProviderID for a community root authority; it must  
1676 be published in a self-describing XRDS document (or other equivalent description—see sections  
1677 9.1.5 and 9.1.6) and verified independently. Once the community root XRDS document is known,  
1678 the ProviderID for delegated XRI authorities within this community MAY be discovered using the  
1679 `xrd:XRD/xrd:Service/xrd:ProviderID` element of authority resolution service endpoints.  
1680 This trust mechanism MAY also be used for other services offered by an authority.

1681 In addition, the metadata necessary for SAML trusted authority resolution or other SAML **[SAML]**  
1682 interactions MAY be discovered using the `ds:KeyInfo` element (section 4.2.) Again, if this  
1683 element is present in an XRD advertising SAML authority resolution service (or any other  
1684 service), and the client has reason to trust this XRD, the client MAY use the associated  
1685 ProviderID to correlate the contents of this element with a signed response.

1686 To assist resolvers in using this key discovery mechanism, it is important that trusted authority  
1687 servers be configured to sign responses in such a way that the signature can be verified using the  
1688 correlated `ds:KeyInfo` element. For more information, see **[SAML]**.

1689 **10.3 HTTPS+SAML**

1690 **10.3.1 Service Type and Service Media Type**

1691 The protocol defined in this section is identified by the values in Table 17.

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

1692 *Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.*

1693 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST  
1694 use the Service Type identifier and Service Media Type identifier defined in Table 17 (including  
1695 the `https=true` and `saml=true` subparameters). In addition, the identifier authority MUST use  
1696 an HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1697 **10.3.2 Protocol**

1698 Following are the normative requirements for HTTPS+SAML trusted authority resolution.

- 1699 1. All authority resolution service endpoints MUST be selected using the values defined in  
1700 Table 17.

- 1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710
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.

## 1711 11 Proxy Resolution Service

1712 The preceding sections have defined XRI resolution as a set of logical functions. This section  
1713 defines a mapping of these functions to an HTTP(S) interface for remote invocation. This  
1714 mapping is based on a standard syntax for expressing an XRI as an HTTP URI, called an *HXRI*,  
1715 as defined in section 11.2. HXRIs also enable XRI resolution input parameters to be encoded as  
1716 query parameters in the HXRI.

1717 Proxy resolution is useful for:

- 1718 • Offloading XRI resolution and service endpoint selection processing from a client to an  
1719 HTTP(S) server.
- 1720 • Optimizing XRD caching for a resolution community (a *caching proxy resolver*). Proxy  
1721 resolvers SHOULD use caching to resolve the same QXRIs or QXRI components for multiple  
1722 clients as defined in section 16.4.
- 1723 • Returning HTTP(S) redirects to clients such as browsers that have no native understanding  
1724 of XRIs but can process HXRIs. This provides backwards compatability with the large  
1725 installed base of existing HTTP clients.

### 1726 11.1 Service Type and Media Types

1727 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

1728 Table 18: Service Type and Service Media Type values for proxy resolution.

1729 A proxy resolution service endpoint advertised in an XRDS document MUST use the Service  
1730 Type identifier and Service Media Type identifiers defined in Table 18. In addition:

- 1731 • An HTTPS proxy resolver MUST specify the media type parameter `https=true` and MUST  
1732 offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service  
1733 endpoint.
- 1734 • A SAML proxy resolver MUST specify the media type parameter `saml=true` and SHOULD  
1735 offer at least one HTTPS URI as the value of the `xrd:URI` element(s) for this service  
1736 endpoint.

1737 It may appear to be of limited value to advertise proxy resolution service in an XRDS document if  
1738 a resolver must already know how to perform local XRI resolution in order to retrieve this  
1739 document. However, advertising a proxy resolution service in the XRDS document for a  
1740 community root authority (sections 9.1.3 and 9.1.6) can be very useful for applications that need  
1741 to consume XRI proxy resolution services or automatically generate HXRIs for resolution by non-  
1742 XRI-aware clients in that community. Those applications may discover the current URI(s) and  
1743 resolution capabilities of a proxy resolver from this source.

### 1744 11.2 HXRIs

1745 The first step in an HTTP binding of the XRI resolution interface is to specify how the QXRI  
1746 parameter is passed within an HTTP(S) URI. Besides providing a binding for proxy resolution,  
1747 defining a standard syntax for expressing an XRI as an HTTP XRI (HXRI) has two other benefits:

1748 • It allows XRIs to be used anywhere an HTTP URI can appear, including in Web pages,  
1749 electronic documents, email messages, instant messages, etc.

1750 • It allows XRI-aware processors and search agents to recognize an HXRI and extract the  
1751 embedded XRI for direct resolution, processing, and indexing.

1752 To make this syntax as simple as possible for XRI-aware processors or search agents to  
1753 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority component that  
1754 begins with the domain name segment "xri.". The QXRI is then appended as the entire local  
1755 path (and query component, if present). The QXRI MUST NOT include the "xri:/" prefix and  
1756 MUST be in URI-normal form as defined in [XRISyntax]. (If a proxy resolver receives an HXRI  
1757 containing a QXRI beginning with an "xri:/" prefix, it SHOULD follow Postel's Law<sup>1</sup> by  
1758 removing it before continuing.) In essence, the proxy resolver URI (including the forward slash  
1759 after the domain name) serves as a machine-readable alternate prefix for an absolute XRI in URI-  
1760 normal form.

1761 The normative ABNF for an HXRI is defined below based on the *ireg-name*, *xri-hier-part*,  
1762 and *iquery* productions defined in [XRISyntax]. XRIs that need to be understood by non-XRI-  
1763 aware clients SHOULD be published as HTTP URIs conforming to this HXRI production.

1764 `HXRI = proxy-resolver "/" QXRI`

---

<sup>1</sup> [http://en.wikipedia.org/wiki/Postel%27s\\_Law](http://en.wikipedia.org/wiki/Postel%27s_Law)

```

1765 proxy-resolver = ( "http://" / "https://" ) proxy-reg-name
1766 proxy-reg-name = "xri." ireg-name
1767 QXRI          = xri-hier-part [ "?" i-query ]

```

1768 URI processors that recognize XRIs SHOULD interpret the local part of an HTTP or HTTPS URI  
 1769 (the path segment(s) and optional query segment) as an XRI provided that: a) it conforms to this  
 1770 ABNF, and b) the first segment of the path conforms to the xri-authority or iauthority productions  
 1771 in [XRISyntax].

1772 For references to communities that offer public XRI proxy resolution services, see the Wikipedia  
 1773 entry on XRI [WikipediaXRI].

### 1774 11.3 HXRI Query Parameters

1775 In proxy resolution, the XRI resolution input parameters defined in section 8.1 are bound to an  
 1776 HTTP(S) interface using the conventional web model of encoding them in an HTTP(S) URI, which  
 1777 in this case is an HXRI. The binding of the logical parameter names to HXRI component parts is  
 1778 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

1779 Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.

1780 Following are the rules for the use of the parameters specified in Table 19.

- 1781 1. The QXRI MUST be normalized as specified in section 11.2.
- 1782 2. If the original QXRI has an existing query component, the HXRI query parameters MUST  
 1783 be appended to that query component.

1784 **IMPORTANT:** The query parameter names in Table 19 were chosen to minimize the probability of  
 1785 collision with any pre-existing query parameter names in the QXRI. If there is any conflict, the  
 1786 pre-existing query parameter names MUST be percent-encoded prior to transformation into an  
 1787 HXRI.

- 1788 3. After proxy resolution, the HXRI query parameters MUST subsequently be removed from  
 1789 the QXRI query component. The existing QXRI query component MUST NOT be altered  
 1790 in any other way, i.e., it must be passed through with no changes in parameter order,  
 1791 escape encoding, etc.
- 1792 4. If the original QXRI does not have a query component, one MUST be added to pass any  
 1793 HXRI query parameters. After proxy resolution, this query component MUST be entirely  
 1794 removed.
- 1795 5. If the original QXRI had a null query component (only a leading question mark), or a  
 1796 query component consisting of only question marks, *one additional leading question mark*  
 1797 MUST be added before adding any HXRI query parameters. After proxy resolution, any

- 1798 HXRI query parameters and exactly one leading question mark MUST be removed. See  
1799 the URI construction steps defined in section 13.5.
- 1800 6. Each HXRI query parameter MUST be delimited from other parameters by an ampersand  
1801 (“&”).
- 1802 7. Each HXRI query parameter MUST be delimited from its value by an equals sign (“=”).
- 1803 8. If an HXRI query parameter includes one of the media type parameters defined in Table  
1804 6, it MUST be delimited from the HXRI query parameter with a semicolon (“;”).
- 1805 9. The fully-composed HXRI MUST be encoded and decoded as specified in section 11.4.
- 1806 10. If any HXRI query parameter name is included but its value is empty, the value of the  
1807 parameter MUST be considered null.

## 1808 11.4 HXRI Encoding/Decoding Rules

1809 To conform with the typical requirements of web server URI parsing libraries, HXRIs MUST be  
1810 encoded prior to input to a proxy resolver and decoded prior to output from a proxy resolver.  
1811 Because web server libraries typically perform some of these decoding functions automatically,  
1812 implementers MUST ensure that a proxy resolver, when used in conjunction with a specific web  
1813 server, accomplishes the full set of HXRI decoding steps specified in this section. In particular,  
1814 these decoding steps MUST be performed prior to any comparison operations defined in this  
1815 specification.

1816 Before any HXRI-specific encoding steps are performed, the QXRI portion of the HXRI (including  
1817 all HXRI query parameters) MUST be transformed into URI-normal form as defined in section 2.3  
1818 of [XRISyntax]. This means characters not allowed in URIs, such as SPACE, or characters that  
1819 are valid only in IRIs, such as UCS characters above the ASCII range, MUST be percent  
1820 encoded. Also, the plus sign character (“+”) MUST NOT be used to encode the SPACE character  
1821 because in decoding the percent-encoded sequence %2B MUST be interpreted as the plus sign  
1822 character (“+”).

1823 Once the HXRI is in URI-normal form, the following sequence of encoding steps MUST be  
1824 performed in the order specified before an HXRI is submitted to a proxy resolver.

1825 **IMPORTANT:** this sequence of steps is not idempotent, so it MUST be performed only once.

- 1826 1. First, in order to preserve percent-encoding when the HXRI is passed through a web  
1827 server, all percent signs MUST be themselves percent-encoded, i.e., a SPACE encoded  
1828 as %20 will become %2520.
- 1829 2. Second, to prevent misinterpretation of HXRI query parameters, any occurrences of the  
1830 ampersand character (“&”) within an HXRI query parameter that are NOT used to delimit  
1831 it from another query parameter MUST be percent encoded using the sequence %26.
- 1832 3. Third, to prevent misinterpretation of the semicolon character by the web server, any  
1833 semicolon used to delimit one of the media type parameters defined in Table 6 from the  
1834 media type value MUST be percent-encoded using the sequence %3B.

1835 To decode an encoded HXRI back into URI-normal form, the above sequence of steps MUST be  
1836 performed in reverse order. Again, the sequence is not idempotent so it MUST be performed only  
1837 once.

1838 Table 20 illustrates the components of an example HXRI before transformation to URI-normal  
1839 form. The characters requiring percent encoding are highlighted in **red**. Note the space in the  
1840 string `hello planète`. Also, for purposes of illustration, the Type component contains a query  
1841 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_t	_xrd_t=http://example.org/test?a=1&b=hello planète
_xrd_m	_xrd_m=application/atom+xml

1842 Table 20: Example of HXRI components prior to transformation to URI-normal form.

1843 Table 21 illustrates these components after transformation to URI-normal form. Characters that  
 1844 have been percent-encoded are in **blue**. Characters still requiring percent encoding according to  
 1845 the rules defined in this section are highlighted in **red**.

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

1846 Table 21: Example of HXRI components after transformation to URI-normal form.

1847 Table 22 illustrates the components after all encoding rules defined in this section are applied.

QXRI	https://xri.example.com/=example*r% <b>25E9</b> sumé% <b>25E9</b> /path?query
_xrd_r	_xrd_r=application/xrds+xml% <b>3B</b> https=true% <b>3B</b> sep=true
_xrd_t	_xrd_t=http://example.org/test?a=1% <b>26</b> b=hello% <b>2520</b> plan% <b>25E8</b> te
_xrd_m	_xrd_m=application/atom+xml

1848 Table 22: Example of HXRI components after application of the required encoding rules.

1849 Following is the fully-encoded HXRI:

```
1850 https://xri.example.com/=example*r%25E9sumé%25E9/path?query
1851 &_xrd_r=application/xrds+xml%3Bhttps=true%3Bsep=true
1852 &_xrd_t=http://example.org/test?a=1%26b=hello%2520plan%25E8te
1853 &_xrd_m=application/atom+xml
```

1854 Following is the fully decoded HXRI returned to URI-normal form. Note that the proxy resolver  
 1855 MUST leave the HXRI in URI-normal form for any further processing.

```
1856 https://xri.example.com/=example*r%E9sumé%E9/path?query
1857 &_xrd_r=application/xrds+xml;https=true;sep=true
1858 &_xrd_t=http://example.org/test?a=1&b=hello%20plan%E8te
1859 &_xrd_m=application/atom+xml
```

## 1860 11.5 HTTP(S) Accept Headers

1861 In proxy resolution, one XRI resolution input parameter, the Service Media Type (section 8.1.4)  
 1862 MAY be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. The  
 1863 following rules apply to this input:

- 1864 1. As described in section 14.1 of **[RFC2616]**, the Accept header content type MAY consist  
 1865 of multiple media type identifiers. If so, the proxy resolver MUST choose only one to

- 1866 accept. A proxy resolver client SHOULD order media type identifiers according to the  
1867 client's preference and a proxy resolver server SHOULD choose the client's highest  
1868 preference.
- 1869 2. If the value of the Accept header content type is null, this MUST be interpreted as the  
1870 value of the Service Media Type parameter.
  - 1871 3. If the value of the Service Media Type parameter is explicitly set via the `_xrd_m` query  
1872 parameter in the HXRI (including to a null value), this MUST take precedence over any  
1873 value set via an HTTP(S) Accept header.

## 1874 11.6 Null Resolution Output Format

1875 Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a  
1876 resolution request where the Resolution Output Format input parameter value is null—either  
1877 because this parameter is absent or because it was explicitly set to null using the `_xrd_r` query  
1878 parameter.

1879 If the value of the Resolution Output Format value is null, a resolver MUST proceed as if the  
1880 following media type parameters had the following values: `https=false`, `saml=false`,  
1881 `refs=true`, `sep=true`, `nodefault_t=false`, `nodefault_p=false`,  
1882 `nodefault_m=false`, and `uric=false`. In addition, the output MUST be an HTTP(S) redirect  
1883 as defined in the following section.

## 1884 11.7 Outputs and HTTP(S) Redirects

1885 For all values of the Resolution Output Format parameter except null, a proxy resolver MUST  
1886 follow the output rules defined in section 8.2.

1887 If the value of the Resolution Output Format is null, and the output is not an error, a proxy  
1888 resolver MUST follow the rules for output of a URI List as defined in section 8.2.3. However,  
1889 instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as  
1890 an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service  
1891 Media Type parameter.

1892 If the output is an error, a proxy resolver SHOULD return a human-readable error message as  
1893 specified in section 15.4.

1894 These rules enable XRI proxy resolvers to serve clients that do not understand XRI syntax or  
1895 resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the  
1896 service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept  
1897 header (if any).

## 1898 11.8 Differences Between Proxy Resolution Servers

1899 An XRI proxy resolution request MAY be sent to any proxy resolver that will accept it. All XRI  
1900 proxy resolvers SHOULD deliver uniform responses given the same QXRI and other input  
1901 parameters. However, because proxy resolvers may potentially need to make decisions about  
1902 network errors, Redirect and Ref processing, and trust policies on behalf of the client they are  
1903 proxying, and these decisions may be based on local policy, in some cases different proxy  
1904 resolvers may return different results.

## 1905 11.9 Combining Authority and Proxy Resolution Servers

1906 The majority of DNS nameservers are recursing nameservers that answer both queries for which  
1907 they are authoritative and queries which they must forward to other nameservers. The same rule  
1908 applies in XRI architecture: in many cases the optimum configuration will be combining an  
1909 authority server and proxy resolver in the same server. This server can publish a self-describing  
1910 XRDS document (section 9.1.6) that advertises both its authority resolution and proxy resolution

1911 service endpoints. It can also optimize caching of XRDs for clients in its resolution community  
1912 (see section 16.4).

1913

## 12 Redirect and Ref Processing

1914  
1915  
1916

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:

1917  
1918  
1919

- 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.

1920  
1921  
1922  
1923

- 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.

1924  
1925

Table 23 summarizes the similarities and differences between the `xrd:Redirect` and `xrd:Ref` elements.

Requirement	Redirect	Ref
Must contain	HTTP(S) URI	XRI
Accepts the same <code>append</code> attribute as the <code>xrd:URI</code> element	Yes	No
Delegates to a different identifier authority	No	Yes
Must include a subset of the synonyms available in the source XRD	Yes	No
Available at both XRD level and SEP level	Yes	Yes
Processed automatically if present at the XRD level	Yes	Yes
Always results in nested XRDS document, even if only to report an error	Yes	Yes
Required attribute of XRDS element for nested XRDS document	<code>redirect</code>	<code>ref</code>
Number of XRDs in nested XRDS document	1	1 or more

1926

Table 23: Comparison of Redirect and Ref elements.

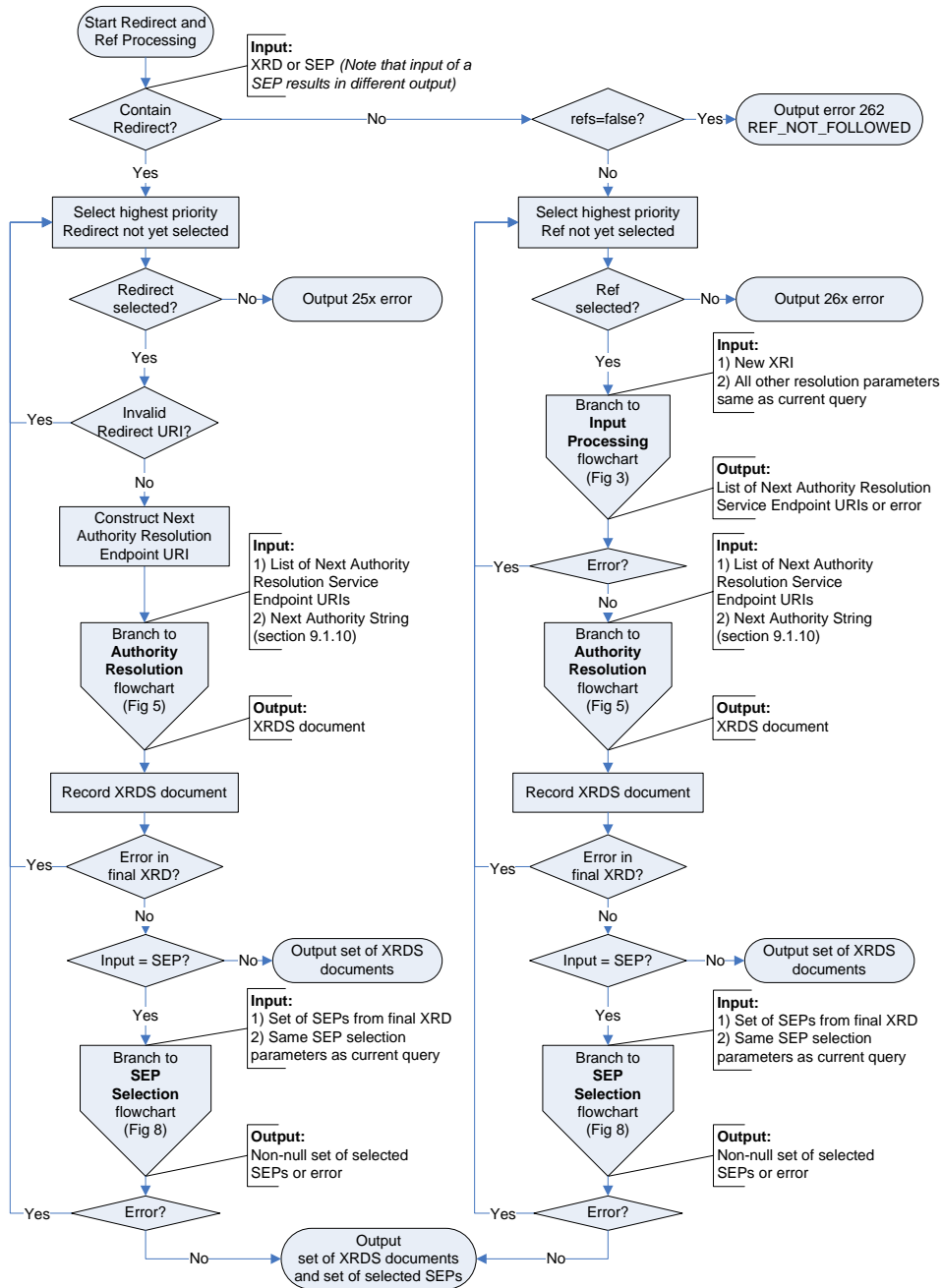
1927  
1928

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

1929  
1930  
1931  
1932  
1933

**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*.

1934 Figure 7 (non-normative) illustrates the logical flow of Redirect and Ref processing.



1935

1936 Figure 7: Redirect and Ref processing flowchart.

1937 This section contains the normative requirements for processing of `xrd:Redirect` and  
1938 `xrd:Ref` elements.

## 1939 12.1 Cardinality

1940 Redirect and Ref elements may be used both at the XRD level (as a child of the `xrd:XRD`  
1941 element) and the SEP level (as a child of the `xrd:XRD/xrd:Service` element) within an XRD.  
1942 In both cases, to simplify processing, the XRD schema (Appendix B) enforces the following rules:

- 1943 • At the XRD level, an XRD MUST contain only one of two choices: zero-or-more  
1944 `xrd:Redirect` or zero-or-more `xrd:Ref` elements.
- 1945 • At the SEP level, a SEP MUST contain only one of three choices: zero-or-more `xrd:URI`  
1946 elements, zero-or-more `xrd:Redirect` elements, or zero-or-more `xrd:Ref` elements.

## 1947 12.2 Precedence

1948 XRDS authors should take special note of the following precedence rules for Redirect and Refs.

- 1949 1. If a Redirect or Ref element is present at the XRD level, it MUST be processed  
1950 immediately before a resolver continues with authority resolution, performs service  
1951 endpoint selection (required or optional), or returns its final output. This rule applies  
1952 recursively to all XRDS documents resolved as a result of Redirect or Ref processing.
- 1953 2. If a Redirect or Ref element is not present at the XRD level, but is present in the highest  
1954 priority service endpoint selected by the rules in section 13, it MUST be processed  
1955 immediately before a resolver completes service endpoint selection (required or optional),  
1956 or returns its final output. This rule also applies recursively to all XRDS documents  
1957 resolved as a result of Redirect or Ref processing.

1958 **IMPORTANT:** Due to these rules, even if a resolver has resolved the final subsegment of an XRI,  
1959 the authority resolution phase is still not complete as long as the final XRD has a Redirect or Ref  
1960 at the XRD level. This Redirect or Ref MUST be resolved until it returns an XRD that does not  
1961 contain an Redirect or Ref at the XRD level. The same rule applies to the optional service  
1962 endpoint selection phase: it is not complete until it locates a final XRD that contains the requested  
1963 SEP but: a) the XRD does not contain an Redirect or Ref at the XRD level, and b) the highest  
1964 priority selected SEP does not contain a Redirect or Ref.

1965 Based on these rules, the following best practices are recommended.

- 1966 1. XRDS authors SHOULD NOT put any service endpoints in an XRD that contains a  
1967 Redirect or Ref at the XRD level because by definition these service endpoints will be  
1968 ignored.
- 1969 2. XRDS authors SHOULD use a Redirect or Ref element at the XRD level if they wish to  
1970 relocate or delegate resolution behavior regardless of any service endpoint query.
- 1971 3. XRDS authors SHOULD use a Redirect or Ref element in a service endpoint for which  
1972 they expect a POSITIVE match as defined in section 13.3.1 if they wish to control  
1973 resolution behavior based on an explicit service endpoint match.
- 1974 4. XRDS authors SHOULD use a Redirect or Ref element in a service endpoint for which  
1975 they expect a DEFAULT match as defined in section 13.3.1 if they wish to control  
1976 resolution behavior based on the absence of an explicit service endpoint match.
- 1977 5. XRDS authors SHOULD NOT include two or more SEPs of equal priority in an XRD if  
1978 they contain Redirects or Refs that will make resolution ambiguous or non-deterministic.

1979 Also note that, during the authority resolution phase, a Redirect or Ref placed in the authority  
1980 resolution SEP of an XRD will have effectively the same result as a Redirect or Ref placed at the  
1981 XRD level. The first option SHOULD be used if the XRD contains other service endpoints or

1982 metadata describing the resource. The second option SHOULD be used only if the XRD contains  
1983 no service endpoints.

## 1984 12.3 Redirect Processing

1985 The purpose of the `xrd:Redirect` element is to enable an authority to redirect from an XRDS  
1986 document managed in one network location (e.g., a registry) to a different XRDS document  
1987 managed in a different network location by the same authority (e.g., a web server, blog, etc.) It is  
1988 similar to an HTTP(S) redirect; however, it is managed at the XRDS document level rather than  
1989 HTTP(S) transport level. Note that unlike a `Ref`, a `Redirect` does NOT delegate to a different XRI  
1990 authority, but only to the same authority at a different network location.

1991 Following are the normative rules for processing of the `xrd:Redirect` element.

- 1992 1. To process a `Redirect` at either the XRD or SEP level, the resolver MUST begin by  
1993 selecting the highest priority `xrd:XRD/xrd:Redirect` element in the XRD or SEP.
- 1994 2. If the value of the resolution subparameter `https` is `FALSE`, or the subparameter is  
1995 absent or empty, the value of the selected `xrd:Redirect` element MUST be EITHER a  
1996 valid HTTP URI or a valid HTTPS URI. If not, the resolver MUST select the next highest  
1997 priority `xrd:Redirect` element. If all instances of this element fail, the resolver MUST  
1998 stop and return the error 251 `INVALID_REDIRECT` in the XRD containing the `Redirect`  
1999 or as a plain text error message as specified in section 15.
- 2000 3. If the value of the resolution subparameter `https` is `TRUE`, the value of the selected  
2001 `xrd:Redirect` element MUST be a valid HTTPS URI. If not, the resolver MUST select  
2002 the next highest priority `xrd:Redirect` element. If all instances of this element fail, the  
2003 resolver MUST stop and return the error 252 `INVALID_HTTPS_REDIRECT` in the XRD  
2004 containing the `Redirect` or as a plain text error message as specified in section 15.
- 2005 4. Once a valid `xrd:Redirect` element has been selected, if the  
2006 `xrd:XRD/xrd:Redirect` element includes the `append` attribute, the resolver MUST  
2007 construct the final HTTP(S) URI as defined in section 13.6.
- 2008 5. The resolver MUST request a new XRDS document from the final HTTP(S) URI using the  
2009 protocol defined in section 6.3. If the Resolution Output Format is an XRDS document,  
2010 the resolver MUST embed a nested XRDS document containing an XRD representing  
2011 the `Redirect` as specified in section 12.5.
- 2012 6. If resolution of an `xrd:Redirect` element fails during the authority resolution phase of  
2013 the original resolution query, or if resolution of an `xrd:Redirect` element fails during  
2014 the optional service endpoint selection phase OR the final XRD does not contain the  
2015 requested SEP, then the resolver MUST report the error in the final XRD of the nested  
2016 XRDS document using the status codes defined in section 15. (One nested XRDS  
2017 document will be added for each `Redirect` attempted by the resolver.) The resolver MUST  
2018 then select the next highest priority `xrd:Redirect` element from the original XRD or  
2019 SEP and repeat rule 7. For more details, see section 12.6, *Recursion and Backtracking*.
- 2020 7. If resolution of all `xrd:Redirect` elements in the XRD or SEP that originally triggered  
2021 `Redirect` processing fails, the resolver MUST stop and return a 25x error in the XRD  
2022 containing the `Redirect` or as a plain text error message as specified in section 15. The  
2023 resolver MUST NOT try any other SEPs even if multiple SEPs were selected as specified  
2024 in section 13.
- 2025 8. If resolution succeeds, the resolver MUST verify the synonym elements in the new XRD  
2026 as specified in section 14.1. If synonym verification fails, the resolver MUST stop and  
2027 return the error specified in that section.
- 2028 9. If the value of the resolution subparameter `saml` is `TRUE`, the resolver MUST verify the  
2029 signature on the XRD as specified in section 10.2.4. If signature verification fails, the  
2030 resolver MUST stop and return the error specified in that section.

2031 10. If Redirect resolution succeeds, further authority resolution or service endpoint selection  
2032 MUST continue based on the new XRD.

## 2033 12.4 Ref Processing

2034 The purpose of the `xrd:Redirect` element is to enable one authority to delegate management  
2035 of all or part of an XRDS document to another authority. For example, an individual might  
2036 delegate management of all or portions of an XRDS document to his/her spouse, school,  
2037 employer, doctor, etc. This delegation may cover the entire document (an XRD level Ref), or only  
2038 one or more specific service endpoints within the document (a SEP level Ref).

2039 Following are the normative rules for processing of the `xrd:Ref` element.

- 2040 1. Ref processing is only be performed if the value of the `refs` subparameter (Table 6) is  
2041 TRUE or it is absent or empty. If the value is FALSE and the XRD contains at least one  
2042 `xrd:Ref` element that could be followed to complete the resolution query, the resolver  
2043 MUST stop and return the error 262 REF\_NOT\_FOLLOWED in the XRD containing the  
2044 Ref or as a plain text error message as defined in section 15. The rules below presume  
2045 that `refs=true`.
- 2046 2. To process a Ref at either the XRD or SEP level, the resolver MUST begin by selecting  
2047 the highest priority `xrd:XRD/xrd:Ref` element from the XRD or SEP.
- 2048 3. The value of the selected `xrd:Ref` element MUST be a valid absolute XRI. If not, the  
2049 resolver MUST select the next highest priority `xrd:Ref` element. If all instances of this  
2050 element fail, the resolver MUST stop and return the error 261 INVALID\_REF in the XRD  
2051 containing the Ref or as a plain text error message as defined in section 15.
- 2052 4. Once a valid `xrd:XRD/xrd:Ref` value is selected, the resolver MUST begin resolution  
2053 of a new XRDS document from this XRI using the protocols defined in this specification.  
2054 Other than the QXRI, the resolver MUST use the same resolution query parameters as  
2055 the original query. If the Resolution Output Format is an XRDS document, the resolver  
2056 MUST embed a nested XRDS document containing an XRD representing the Ref as  
2057 defined in section 12.5.
- 2058 5. If resolution of an `xrd:Ref` element fails during the authority resolution phase of the  
2059 original resolution query, or if resolution of an `xrd:Ref` element fails during the optional  
2060 service endpoint selection phase OR the final XRD does not contain the requested  
2061 service endpoint, then the resolver MUST record the nested XRDS document as far as  
2062 resolution was successful, including the relevant status codes for each XRD as specified  
2063 in section 15. The resolver MUST then select the next highest priority `xrd:Ref` element  
2064 as specified above and repeat rule 5. For more details, see section 12.6, *Recursion and*  
2065 *Backtracking*.
- 2066 6. If resolution of all `xrd:Ref` elements in the XRD or SEP originating Ref processing fails,  
2067 the resolver MUST stop and return a 26x error in the XRD containing the Ref or as a  
2068 plain text error message as specified in section 15. The resolver MUST NOT try any  
2069 other SEPs even if multiple SEPs were selected as specified in section 13.
- 2070 7. If resolution of an `xrd:Ref` element succeeds and `cid=true`, the resolver MUST  
2071 perform CanonicalID verification across all XRDs in the nested XRDS document as  
2072 specified in section 14.3. Note that each set of XRDs in each new nested XRDS  
2073 document produced as a result of Redirect or Ref processing constitutes its own  
2074 CanonicalID verification chain. *CanonicalID verification never crosses between XRDS*  
2075 *documents*. See section 12.5 for examples.
- 2076 8. If resolution of an `xrd:Ref` element succeeds and the final XRD contains the service  
2077 endpoint(s) necessary to continue or complete the original resolution query, further  
2078 authority resolution or service endpoint selection MUST continue based on the final XRD.

## 2079 12.5 Nested XRDS Documents

2080 Processing of a Redirect or Ref ALWAYS produces a new XRDS document that describes the  
2081 Redirect or Ref that was followed, even if the result was an error. If the final requested Resolution  
2082 Output Format is NOT an XRDS document, this new XRDS document is only needed to obtain  
2083 the metadata necessary to continue or complete resolution. However, if the final requested  
2084 Resolution Output Format is an XRDS document, each XRDS document produced as a result of  
2085 Redirect or Ref processing MUST be nested inside the outer XRDS document immediately  
2086 following the `xrd:XRD` element containing the `xrd:Redirect` or `xrd:Ref` element being  
2087 followed. If more than one Redirect or Ref element is resolved due to an error, the corresponding  
2088 nested XRDS documents MUST be included in the same order as the Redirect or Ref elements  
2089 that were followed to produce them.

2090 Each new XRDS document is a recursive authority resolution call and MUST conform to all  
2091 authority resolution requirements. In addition, the following rules apply:

- 2092 • For a Redirect, the `xrds:XRDS/@redirect` attribute of the nested XRDS document MUST  
2093 contain the fully-constructed HTTP(S) URI it describes as specified in section 12.3.
- 2094 • For a Ref, the `xrds:XRDS/@ref` attribute of the nested XRDS document MUST contain the  
2095 exact value of the `xrd:XRD/xrd:Ref` element it describes.

2096 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the  
2097 original query identifier even if resolution traverses multiple Redirects or Refs, and even if errors  
2098 were encountered. Note that like the outer XRDS document, nested XRDS documents MUST  
2099 NOT include an XRD for the community root subsegment because this is part of the configuration  
2100 of the resolver.

2101 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD with an  
2102 `xml:id` attribute value matching the `xml:id` attribute value of any previous XRD in the chain of  
2103 resolution requests beginning with the original QXRI, the resolver MUST replace this XRD with an  
2104 empty XRD element. The resolver MUST set this empty element's `idref` attribute value to the  
2105 value of the `xml:id` attribute of the matched XRD element. This prevents conflicting `xml:id`  
2106 values.

### 2107 12.5.1 Redirect Examples

#### 2108 Example #1:

2109 In this example the original query identifier is `xri://@a`. The first XRD contains an XRD-level  
2110 Redirect to `http://a.example.com/`. The elements and attributes specific to Redirect  
2111 processing are shown in **bold**. CanonicalIDs are included to illustrate the synonym verification  
2112 rule in section 12.3.

```
2113 <XRDS xmlns="xri://$xrds" ref="xri://@a">
2114   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2115     <Query>*a</Query>
2116     <ProviderID>xri://@</ProviderID>
2117     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2118     <Redirect>http://a.example.com/</Redirect>
2119     ...
2120   </XRD>
2121   <XRDS redirect="http://a.example.com/">
2122     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2123       <ProviderID>xri://@</ProviderID>
2124       <CanonicalID>xri://@!1</CanonicalID> ;SAME AS XRDS #1 CID #1
2125       ...
2126       <Service>
2127         <Type>http://openid.net/signon/1.0</Type>
2128         <URI>http://openid.example.com/</URI>
2129       </Service>
```

2130  
2131  
2132

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

### 2133 **Example #2:**

2134 In this example the original query identifier is `xri://a*b*c`. The second XRD contains a SEP-  
2135 level Redirect in its authority resolution SEP to `http://other.example.com/`. Note that  
2136 because authority resolution is not complete when this Redirect is encountered, it continues in the  
2137 outer XRDS after the nested XRDS representing the Redirect is complete. Again, CanonicalIDs  
2138 are included to illustrate the synonym verification rule.

2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182

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

### 2183 **Example #3:**

2184 In this example the original query identifier is again `xri://@a*b*c`. This time the final XRD  
 2185 contains a SEP-level Redirect to `http://other.example.com/`. Because authority resolution  
 2186 is complete, the outer XRDS ends with a nested XRDS representing the SEP-level Redirect.

```

2187 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2188   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2189     <Query>*a</Query>
2190     <ProviderID>xri://@</ProviderID>
2191     <CanonicalID>xri://@!1</CanonicalID>           ;XRDS #1 CID #1
2192     ...
2193     <Service>
2194       <Type>xri://$res*auth*($v*2.0)</Type>
2195       <URI>http://a.example.com/</URI>
2196     </Service>
2197   </XRD>
2198   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2199     <Query>*b</Query>
2200     <ProviderID>xri://@!1</ProviderID>
2201     <CanonicalID>xri://@!1!2</CanonicalID>       ;XRDS #1 CID #2
2202     ...
2203     <Service>
2204       <Type>xri://$res*auth*($v*2.0)</Type>
2205       <URI>http://b.example.com/</URI>
2206     </Service>
2207   </XRD>
2208   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2209     <Query>*c</Query>
2210     <ProviderID>xri://@!1!2</ProviderID>
2211     <CanonicalID>xri://@!1!2!3</CanonicalID>    ;XRDS #1 CID #3
2212     ...
2213     <Service>
2214       <Type>http://openid.net/signon/1.0</Type>
2215       <Redirect>http://r.example.com/openid</Redirect>
2216     </Service>
2217   </XRD>
2218   <XRDS redirect="http://r.example.com/openid">
2219     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2220       <ProviderID>xri://@!1!2</ProviderID>
2221       <CanonicalID>xri://@!1!2!3</CanonicalID>  ;SAME AS XRDS #1 CID
2222 #3
2223       ...
2224       <Service>
2225         <Type>http://openid.net/signon/1.0</Type>
2226         <URI>http://openid.example.com/</URI>
2227       </Service>
2228     </XRD>
2229   </XRDS>
2230 </XRDS>

```

## 2231 12.5.2 Ref Examples

### 2232 Example #1:

2233 In this example the original query identifier is `xri://@a`. The first XRD contains an XRD-level  
 2234 Ref to `xri://@x*y`. The CanonicalID values are included to illustrate the CanonicalID  
 2235 verification rules in section 14.3.

```

2236 <XRDS xmlns="xri://$xrds" ref="xri://@a">
2237   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2238     <Query>*a</Query>
2239     <ProviderID>xri://@</ProviderID>
2240     <CanonicalID>xri://@!1</CanonicalID>           ;XRDS #1 CID #1

```

```

2241     <Ref>xri://@x*y</Ref>
2242   </XRD>
2243   <XRDS ref="xri://@x*y">
2244     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2245       <Query>*x</Query>
2246       <ProviderID>xri://@</ProviderID>
2247       <CanonicalID>xri://@!7</CanonicalID> ;XRDS #2 CID #1
2248       ...
2249       <Service>
2250         <Type>xri://$res*auth*($v*2.0)</Type>
2251         <URI>http://x.example.com/</URI>
2252       </Service>
2253     </XRD>
2254     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2255       <Query>*y</Query>
2256       <ProviderID>xri://@!7</ProviderID>
2257       <CanonicalID>xri://@!7!8</CanonicalID> ;XRDS #2 CID #2
2258       ...
2259       <Service>
2260         <Type>xri://$res*auth*($v*2.0)</Type>
2261         <URI>http://y.example.com/</URI>
2262       </Service>
2263       <Service>
2264         <Type>http://openid.net/signon/1.0</Type>
2265         <URI>http://openid.example.com/</URI>
2266       </Service>
2267     </XRD>
2268   </XRDS>
2269 </XRDS>

```

2270 **Example #2:**

2271 In this example the original query identifier is `xri://@a*b*c`. The second XRD contains a SEP-  
2272 level Ref in its authority resolution SEP to `xri://@x*y`. Note that because authority resolution is  
2273 not complete when this Ref is encountered, it continues in the outer XRDS after the nested XRDS  
2274 representing the Ref. *Note especially how the CanonicalIDs progress to satisfy the CanonicalID*  
2275 *verification rules specified in section 14.3.*

```

2276 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2277   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2278     <Query>*a</Query>
2279     <ProviderID>xri://@</ProviderID>
2280     <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2281     ...
2282     <Service>
2283       <Type>xri://$res*auth*($v*2.0)</Type>
2284       <URI>http://a.example.com/</URI>
2285     </Service>
2286   </XRD>
2287   <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2288     <Query>*b</Query>
2289     <ProviderID>xri://@!1</ProviderID>
2290     <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2
2291     ...
2292     <Service>
2293       <Type>xri://$res*auth*($v*2.0)</Type>
2294       <Ref>xri://@x*y</Ref>
2295     </Service>
2296   </XRD>
2297   <XRDS ref="xri://@x*y">
2298     <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2299       <Query>*x</Query>

```

```

2300 <ProviderID>xri://@/</ProviderID>
2301 <CanonicalID>xri://@!7</CanonicalID> ;XRDS #2 CID #1
2302 ...
2303 <Service>
2304 <Type>xri://$res*auth*($v*2.0)</Type>
2305 <URI>http://x.example.com/</URI>
2306 </Service>
2307 </XRD>
2308 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2309 <Query>*y</Query>
2310 <ProviderID>xri://@!7</ProviderID>
2311 <CanonicalID>xri://@!7!8</CanonicalID> ;XRDS #2 CID #2
2312 ...
2313 <Service>
2314 <Type>xri://$res*auth*($v*2.0)</Type>
2315 <URI>http://y.example.com/</URI>
2316 </Service>
2317 </XRD>
2318 </XRDS>
2319 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2320 <Query>*c</Query>
2321 <ProviderID>xri://@!1!2</ProviderID>
2322 <CanonicalID>xri://@!1!2!3</CanonicalID> ;XRDS #1 CID #3 IS
2323 CHILD OF XRDS #1 CID #2
2324 ...
2325 <Service>
2326 ...final service endpoints described here...
2327 </Service>
2328 </XRD>
2329 </XRDS>

```

2330 **Example #3:**

2331 In this example the original query identifier is again `xri://@a*b*c`. This time the final XRD  
2332 contains a SEP-level Ref to `xri://@x*y`. Because authority resolution is complete, the outer  
2333 XRDS ends with a nested XRDS representing the SEP-level Ref.

```

2334 <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2335 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2336 <Query>*a</Query>
2337 <ProviderID>xri://@/</ProviderID>
2338 <CanonicalID>xri://@!1</CanonicalID> ;XRDS #1 CID #1
2339 ...
2340 <Service>
2341 <Type>xri://$res*auth*($v*2.0)</Type>
2342 <URI>http://a.example.com/</URI>
2343 </Service>
2344 </XRD>
2345 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2346 <Query>*b</Query>
2347 <ProviderID>xri://@!1</ProviderID>
2348 <CanonicalID>xri://@!1!2</CanonicalID> ;XRDS #1 CID #2
2349 ...
2350 <Service>
2351 <Type>xri://$res*auth*($v*2.0)</Type>
2352 <URI>http://a.example.com/</URI>
2353 </Service>
2354 </XRD>
2355 <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2356 <Query>*c</Query>
2357 <ProviderID>xri://@!1!2</ProviderID>
2358 <CanonicalID>xri://@!1!2!3</CanonicalID> ;XRDS #1 CID #3

```

2359  
2360  
2361  
2362  
2363  
2364  
2365  
2366  
2367  
2368  
2369  
2370  
2371  
2372  
2373  
2374  
2375  
2376  
2377  
2378  
2379  
2380  
2381  
2382  
2383  
2384  
2385  
2386  
2387  
2388  
2389  
2390  
2391

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

2392 **12.6 Recursion and Backtracking**

2393 Redirect and Ref processing triggers recursive calls to authority resolution that produce nested  
2394 XRDS documents. This recursion can continue to any depth, i.e., a Redirect may contain another  
2395 Redirect or a Ref, and a Ref may contain another Ref or a Redirect. To avoid confusion, either in  
2396 resolver implementations or in XRDS documents, it is important to clarify the “backtracking” rules.  
2397 The following should be read in conjunction with the flowcharts in Figure 2, Figure 5, Figure 7,  
2398 and Figure 8.

- 2399 • *Separation of phases.* Redirect and Ref processing invoked during the authority resolution  
2400 phase is separate and distinct from Redirect and Ref processing invoked during the optional  
2401 service endpoint selection phase (see Figure 2). Redirect or Ref processing during the former  
2402 MUST successfully complete authority resolution or else return an error. Redirect or Ref  
2403 processing during the latter MUST successfully locate the requested service endpoint or else  
2404 return an error, i.e., it MUST NOT backtrack into the authority resolution phase.
- 2405 • *First recursion point.* The first time a resolver encounters a Redirect or a Ref within a phase is  
2406 called the *first recursion point*. There MUST be at most one first recursion point during the  
2407 authority resolution phase and at most one first recursion point during the optional service  
2408 endpoint selection phase. During the authority resolution phase, the first recursion point MAY  
2409 be either an XRD or a service endpoint (SEP). During the optional service endpoint selection  
2410 phase, the first recursion point MUST be a SEP.
- 2411 • *Priority order.* As specified in sections 12.3 and 12.4, once a resolver reaches a first  
2412 recursion point during the authority resolution stage, it MUST process Redirects or Refs in  
2413 priority order until either it successfully completes authority resolution (and the final XRD  
2414 does not contain an XRD-level Redirect or Ref), or until all Redirects or Refs have failed.

2415 Similarly, once a resolver reaches a first recursion point during the optional service endpoint  
2416 selection phase, it MUST process Redirect or Ref in priority order until either it successfully  
2417 locates the requested SEP (and that SEP does not contain a Redirect or Ref), or until all  
2418 Redirects or Refs have failed.

Comment [DSR11]: PROOF –  
Reworded per suggestion from John.

- 2419 • *Next recursion point.* If a Redirect or Ref leads to another Redirect or Ref, this is called the  
2420 *next recursion point*. The same rules apply to the next recursion point as apply to the first  
2421 recursion point, except that if any next recursion point completely fails, the resolver MUST  
2422 return to the previous recursion point and continue trying any untried Redirects or Refs until  
2423 either it is successful or all Redirects or Refs have failed.
- 2424 • *Termination.* If the resolver returns to the first recursion point and all of its Redirects or Refs  
2425 have failed, the resolver MUST stop and return an error.

2426 To avoid excessive recursion and inefficient resolution responses, XRDS authors are  
2427 RECOMMENDED to use as few Redirects or Refs in a resolution chain as possible.

2428

## 13 Service Endpoint Selection

2429

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

2430

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

2431

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

2432

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

2433

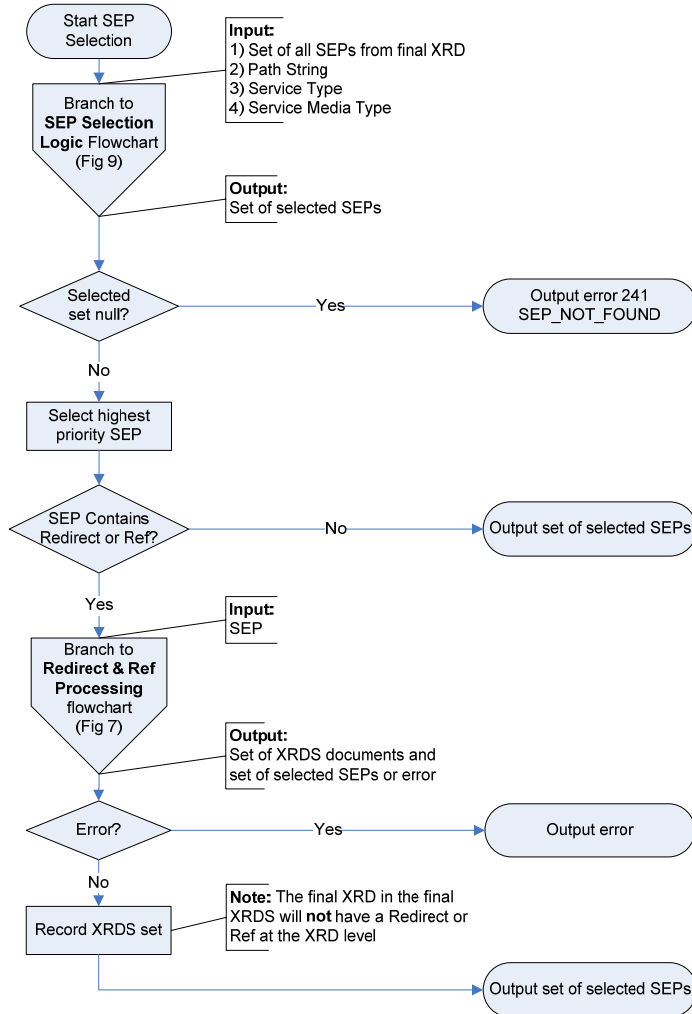
requested.

2434

### 13.1 Processing Rules

2435

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



2436

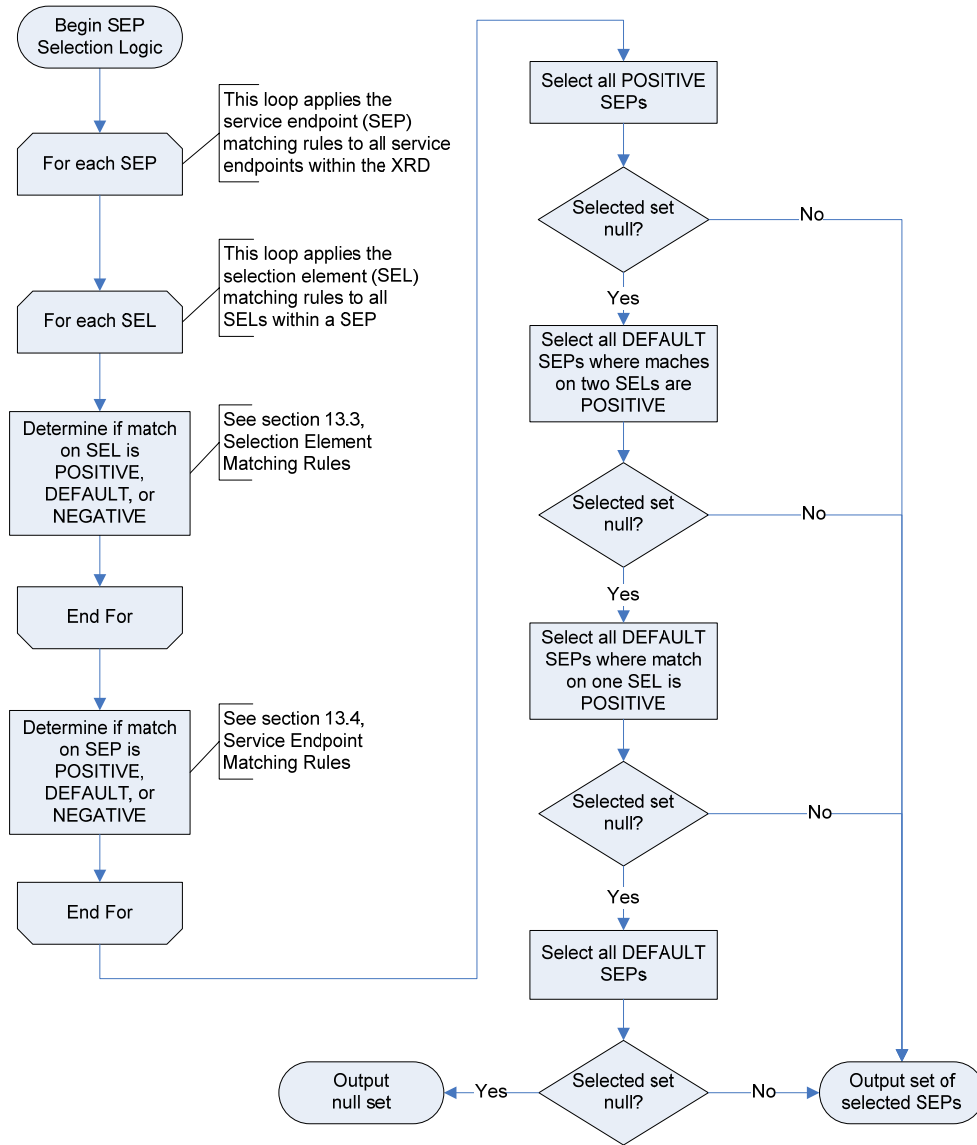
2437

Figure 8: Service endpoint selection flowchart.

- 2438 Following are the normative rules for the overall service endpoint selection process:
- 2439 1. The inputs for service endpoint selection are defined in Table 8.
- 2440 2. For the set of all service endpoints (`xrd:XRD/xrd:Service` elements) in the XRD,  
2441 service endpoint selection MUST follow the logic defined in section 0. The output of this  
2442 process MUST be either the null set or a selected set of one or more service endpoints.
- 2443 3. If, after applying the service endpoint selection logic, the selected set is null, this function  
2444 MUST return the error 241 SEP\_NOT\_FOUND.
- 2445 4. If, after applying the service endpoint selection logic, the selected set is not null and the  
2446 highest priority selected service endpoint contains an  
2447 `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`  
2448 element, it MUST first be processed as specified in section 12. This is a recursive call  
2449 that will produce a nested XRDS document as defined in section 12.5.

2450 **Service Endpoint Selection Logic**

2451 Selection of service endpoints (SEPs) within an XRD is managed using service endpoint  
 2452 selection elements (SEs). As shown in Figure 9 (non-normative), the selection process first  
 2453 applies SEL matching rules (section 13.2), followed by SEP matching rules (section 13.3), to the  
 2454 set of all SEPs in the XRD. It then applies SEP selection rules (section 13.4) to determine the  
 2455 final output.



2456  
 2457 *Figure 9: Service endpoint (SEP) selection logic flowchart.*

2458 The following sections provide the normative rules for each section of this flowchart.

## 2459 13.2 Selection Element Matching Rules

2460 The first set of rules govern the matching of selection elements.

### 2461 13.2.1 Selection Element Match Options

2462 As defined in section 4.2.6, there are three categories of service endpoint selection elements:  
2463 `xrd:Type`, `xrd:Path`, and `xrd:MediaType`. Within each service endpoint, there is a match  
2464 option for each of the three categories of selection elements. Matches are tri-state: the three  
2465 options and their corresponding precedence order are defined in Table 24:

Match Option	Match Condition	Precedence
POSITIVE	A successful match based on the value of the <code>match</code> attribute as defined in 13.2.2 OR a successful match based the contents of the selection element as defined in sections 13.2.6 - 13.2.8.	1
DEFAULT	The value of the <code>match</code> attribute is <code>default</code> OR there is no instance of this type of selection element contained in the service endpoint as defined in section 13.2.3.	0
NEGATIVE	The selection element does not satisfy either condition above.	-1

2466 Table 24: Match options for selection elements.

2467 The Precedence order is used in the Multiple Selection Element Matching Rule (section 13.2.5).

2468 **IMPORTANT:** Failure of a POSITIVE match does not necessarily mean a NEGATIVE match; it  
2469 may still qualify as a DEFAULT match.

### 2470 13.2.2 The Match Attribute

2471 All three service endpoint selection elements accept the optional `match` attribute. This attribute  
2472 gives XRDS authors precise control over selection of SEPs based on the QXRI and other service  
2473 endpoint selection parameters. An enumerated list of the values for the `match` attribute is defined  
2474 in Table 25. If the `match` attribute is present with one of these values, the contents of the  
2475 selection element **MUST** be ignored, and the corresponding matching rule **MUST** be applied. If  
2476 the `match` attribute is absent or has any other value, the rules in this section do not apply.

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

2477 Table 25: Enumerated values of the global match attribute and corresponding matching rules.

2478 BACKWARDS COMPATABILITY NOTE: earlier working drafts of this specification included the  
2479 values `match="none"` and `match="contents"`. Both are deprecated. The former is no longer  
2480 supported and the latter is now the default behaviour of any selection element that does not  
2481 include the `match` attribute. Implementers SHOULD accept these values accordingly.

### 2482 13.2.3 Absent Selection Element Matching Rule

2483 If a service endpoint does not contain at least one instance of a particular category of selection  
2484 element, it MUST be considered equivalent to the service endpoint having a DEFAULT match on  
2485 that category of selection element UNLESS overridden by a `nodefault_*` parameter as specified  
2486 in Table 25.

### 2487 13.2.4 Empty Selection Element Matching Rule

2488 If a selection element is present in a service endpoint but the element is empty, and if the element  
2489 does not contain a `match` attribute, it MUST be considered equivalent to having a `match`  
2490 attribute with a value of `null`.

### 2491 13.2.5 Multiple Selection Element Matching Rule

2492 Each service endpoint has only one match option for each category of selection element.  
2493 Therefore if a service endpoint contains more than one instance of the same category of selection  
2494 element (i.e., more than one `xrd:Type`, `xrd:Path`, or `xrd:MediaType` element), the match for  
2495 that category of selection element MUST be the match for the selection element(s) with the  
2496 highest precedence match option as defined in Table 24.

### 2497 13.2.6 Type Element Matching Rules

2498 The following rules apply to matching the value of the input Service Type parameter with the  
2499 contents of a non-empty `xrd:XRD/xrd:Service/xrd:Type` element when its `match` attribute  
2500 is absent.

- 2501 1. If the value is an XRI or IRI, it MUST be in URI-normal form as defined in section 4.4.
- 2502 2. Prior to comparison (and only for the purpose of comparison), the values of the Service  
2503 Type parameter and the `xrd:XRD/xrd:Service/xrd:Type` element SHOULD be  
2504 normalized according to the requirements of their identifier scheme. In particular, if an  
2505 XRI, IRI, or URI uses hierarchical syntax and does not include a local part (a path and/or  
2506 query component) after the authority component, a trailing forward slash after the  
2507 authority component MUST NOT be considered significant in comparisons. In all other  
2508 cases, a trailing forward slash MUST be considered significant in comparisons unless this  
2509 rule is overridden by scheme-specific comparison rules.
- 2510 3. To result in a POSITIVE match on this selection element, the values MUST be equivalent  
2511 according to the equivalence rules of the applicable identifier scheme. Any other result is  
2512 a NEGATIVE match on this selection element.

2513 As a best practice, service architects SHOULD assign identifiers for service types that are in URI-  
2514 normal form, do not require further normalization, and are easy to match.

### 2515 13.2.7 Path Element Matching Rules

2516 The following rules apply to matching the value of the input Path String (the path portion of the  
2517 QXRI as defined in section 8.1.1) with the contents of a non-empty  
2518 `xrd:XRD/xrd:Service/xrd:Path` element when its `match` attribute is absent.

- 2519 1. If the value is a relative XRI or an IRI it MUST be in URI-normal form as defined in  
2520 section 4.4.

- 2521 2. Prior to comparison, the leading forward slash separating an XRI authority component  
2522 from the path component **MUST** be prepended to the Path String. Any subsequent  
2523 forward slash, including trailing forward slashes, **MUST** be significant in comparisons.
- 2524 3. The contents of the `xrd:XRD/xrd:Service/xrd:Path` element **SHOULD** include the  
2525 leading forward slash separating the XRI authority component from the path. If it does  
2526 not, one **MUST** be prepended prior to comparison.
- 2527 4. Equivalence comparison **SHOULD** be performed using Caseless Matching as defined in  
2528 section 3.13 of **[Unicode]**.
- 2529 5. To result in a **POSITIVE** match on this selection element, the value of the Path String  
2530 **MUST** be a *subsegment stem match* with the contents of the  
2531 `xrd:XRD/xrd:Service/xrd:Path` element. A subsegment stem match is defined as  
2532 the entire Path String being character-for-character equivalent with any continuous  
2533 sequence of subsegments or segments (including empty subsegments and empty  
2534 segments) in the contents of the Path element beginning from the most significant  
2535 (leftmost) subsegment. Subsegments and segments are formally defined in **[XRISyntax]**.  
2536 Any other result **MUST** be a **NEGATIVE** match on this selection element.

2537 Examples of this rule are shown in Table 26.

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

2538 Table 26: Examples of applying the Path element matching rules.

## 2539 13.2.8 MediaType Element Matching Rules

2540 The following rules apply to matching the value of the input Service Media Type parameter with  
2541 the contents of of a non-empty `xrd:XRD/xrd:Service/xrd:MediaType` element when its  
2542 `match` attribute is absent.

- 2543 1. The values of the Service Media Type parameter and the `xrd:MediaType` element  
2544 SHOULD be normalized according to the rules for media types in section 3.7 of  
2545 [RFC2616] prior to input. (The rules are that media type and media type parameter  
2546 names are case-insensitive, but parameter values may or may not be case-sensitive  
2547 depending on the semantics of the parameter name. XRI Resolution Output Format  
2548 parameters and subparameters are all case-insensitive.) XRI resolvers MAY perform  
2549 normalization of these values but MUST NOT be required to do so.
- 2550 2. To be a POSITIVE match on this selection element, the values MUST be character-for-  
2551 character equivalent. Any other result is a NEGATIVE match on this selection element.

## 2552 13.3 Service Endpoint Matching Rules

2553 The next set of matching rules govern the matching of service endpoints based on the matches of  
2554 the selection elements they contain.

### 2555 13.3.1 Service Endpoint Match Options

2556 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 13.3.2) or the All Positive Match Rule (section 13.3.3).
DEFAULT	Meets the Default Match Rule (section 13.3.4).
NEGATIVE	The service endpoint does not satisfy either condition above.

2557 *Table 27: Match options for service endpoints.*

### 2558 13.3.2 Select Attribute Match Rule

2559 All three service endpoint selection elements accept the optional `select` attribute. This attribute  
2560 is a Boolean value used to govern matching of the containing service endpoint according to the  
2561 following rule. If service endpoint contains a selection element with a POSITIVE match as defined  
2562 in section 13.2, and the value of this selection element's `select` attribute is TRUE, the service  
2563 endpoint automatically MUST be a POSITIVE match, i.e., all other selection elements for this  
2564 service endpoint MUST be ignored.

### 2565 13.3.3 All Positive Match Rule

2566 If a service endpoint has a POSITIVE match on all three categories of selection elements  
2567 (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) as defined in section 13.2, the service endpoint  
2568 MUST be a POSITIVE match. If even one of the three selection element match types is not  
2569 POSITIVE, this rule fails.

### 2570 13.3.4 Default Match Rule

2571 If a service endpoint fails the Select Attribute Match Rule and the All Positive Match Rule, but  
2572 none of the three categories of selection elements has a NEGATIVE match as defined in section  
2573 13.2, the service endpoint MUST be a DEFAULT match.

## 2574 13.4 Service Endpoint Selection Rules

2575 The final set of rules governs the selection of service endpoints based on their matches.

### 2576 13.4.1 Positive Match Rule

2577 After applying the matching rules to service endpoints in section 13.3, all service endpoints that  
2578 have a POSITIVE match MUST be selected. Only if there are no service endpoints with a  
2579 POSITIVE match is the Default Match Rule invoked.

### 2580 13.4.2 Default Match Rule

2581 If the Positive Match Rule above fails, then the service endpoints with a DEFAULT match that  
2582 have the highest number of POSITIVE matches on each category of selection element MUST be  
2583 selected. This means:

- 2584 1. The service endpoints in the DEFAULT set that have two POSITIVE selection element  
2585 matches MUST be selected.
- 2586 2. If the previous set is empty, the service endpoints in the DEFAULT set that have one  
2587 POSITIVE selection element match MUST be selected.
- 2588 3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.
- 2589 4. If the previous set is empty, no service endpoint is selected and the return set is null.

## 2590 13.5 Pseudocode

2591 The following pseudocode provides a precise description of the service endpoint selection logic.  
2592 The pseudocode is normative, however if there is a conflict between it and the rules stated in the  
2593 preceding sections, the preceding sections shall prevail.

2594 The pseudocode uses nine Boolean flags to record the match state for each category of selection  
2595 element (SEL) in a service endpoint (SEP):

- 2596 • Postive.Type
- 2597 • Postive.Path
- 2598 • Positive.MediaType
- 2599 • Default.Type
- 2600 • Default.Path
- 2601 • Default.MediaType
- 2602 • Present.Type
- 2603 • Present.Path
- 2604 • Present.MediaType

2605 Note that the complete set of nine SEL match flags is needed for each SEP. The pseudocode first  
2606 does a loop through all SEPs in the XRD to:

- 2607 1. Set the SEL match flags according to the rules specified in section 13.2;
- 2608 2. Process the SEL match flags to apply the SEP matching rules specified in section 13.3;
- 2609 3. Apply the positive SEP selection rule specified in section 13.4.1.

2610 After this loop is complete, the pseudocode tests to see if default SEP selection processing is  
2611 required. If so, it performs a second loop applying the default SEP selection rules specified in  
2612 section 13.4.2.

2613

```
2614 FOR EACH SEP
2615     CREATE set of SEL match flags
2616     SET all flags to FALSE
2617     FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
2618         SET Present.x=TRUE
2619         IF match on this SEL is POSITIVE
2620             IF select="true" ;see 12.4.2
2621                 ADD SEP TO SELECTED SET
2622             NEXT SEP
2623         ELSE
2624             SET Positive.x=TRUE
2625         ENDIF
2626     ELSEIF match on this SEL is DEFAULT ;see 10.3.2 & 12.3.4
2627         IF Positive.x != TRUE AND
2628         nodefault != x ;see 12.3.5
2629             SET Default.x=TRUE
2630         ENDIF
2631     ENDIF
2632 ENDFOR
2633 IF Present.x=FALSE ;see 12.3.3
2634     IF nodefault_x != TRUE ;see 10.3.2
2635         SET Default.x=TRUE
2636     ENDIF
2637 ENDIF
2638 IF Positive.Type=TRUE AND
2639     Positive.Path=TRUE AND
2640     Positive.Mediatype=TRUE ;see 12.4.3
2641     ADD SEP TO SELECTED SET
2642     NEXT SEP
2643 ELSEIF SELECTED SET != EMPTY ;see 12.5.1
2644     NEXT SEP
2645 ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
2646     (Positive.Path=TRUE OR Default.Path=TRUE) AND
2647     (Positive.MediaType=TRUE OR Default.MediaType=TRUE)
2648     ADD SEP TO DEFAULT SET ;see 12.4.4
2649 ENDIF
2650 ENDFOR
2651 IF SELECTED SET = EMPTY ;see 12.5.1
2652     FOR EACH SEP IN DEFAULT SET
2653         IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
2654         (Positive.Type=TRUE AND Positive.MediaType=TRUE) OR
2655         (Positive.Path=TRUE AND Positive.MediaType=TRUE)
2656             ADD SEP TO SELECTED SET
2657         ENDIF
2658     ENDFOR
2659 IF SELECTED SET = EMPTY
2660     FOR EACH SEP IN DEFAULT SET
2661         IF Positive.Type=TRUE OR
2662         Positive.Path=TRUE OR
2663         Positive.MediaType=TRUE
2664             ADD SEP TO SELECTED SET
2665         ENDIF
2666     ENDFOR
2667 ENDIF
2668 ENDIF
2669 IF SELECTED SET != EMPTY
2670     RETURN SELECTED SET
2671 ELSE
2672     RETURN DEFAULT SET
2673 ENDIF
```

## 2674 13.6 Construction of Service Endpoint URIs

2675 The final step in the service endpoint selection process is construction of the service endpoint  
2676 URI(s). This step is necessary if either:

- 2677 • The resolution output format is a URI List.
- 2678 • Automatic URI construction is requested using the `uric` parameter.

### 2679 13.6.1 The `append` Attribute

2680 The `append` attribute of a `xrd:XRD/xrd:Service/xrd:URI` element is used to specify how  
2681 the final URI is constructed. The values of this attribute are shown in Table 28.

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

2682 Table 28: Values of the `append` attribute and the corresponding QXRI component to append.

2683 If the `append` attribute is absent, the default value is `none`. Following are the rules for  
2684 construction of the final service endpoint URI based on the value of the `append` attribute.

2685 **IMPORTANT:** Implementers must follow these rules exactly in order to give XRDS authors  
2686 precise control over construction of service endpoint URIs.

- 2687 1. If the value is `none`, the exact contents of the `xrd:URI` element **MUST** be returned  
2688 directly without any further processing.
- 2689 2. For any other value, the exact value in URI-normal form of the QXRI component specified  
2690 in Table 28, *including any leading delimiter(s) and without any additional escaping or*  
2691 *percent encoding* **MUST** be appended directly to the exact contents of the `xrd:URI`  
2692 element *including any trailing delimiter(s)*. If the value of the QXRI component specified in  
2693 Table 28 consists of only a leading delimiter, then this value **MUST** be appended  
2694 according to these rules. If the value of the QXRI component specified in Table 28 is null,  
2695 then the contents of the `xrd:URI` element **MUST** be returned directly exactly as if the  
2696 value of the `append` attribute was `none`.

2697 3. If any HXRI query parameters for proxy resolution were added to an existing QXRI query  
2698 component as defined in section 11.3, these query parameters **MUST** be removed prior  
2699 to performing the append operation as also defined in section 11.3. In particular, if after  
2700 removal of these query parameters the QXRI query component consists of only *a string*  
2701 *of one or more question marks* (the delimiting question mark plus zero or more additional  
2702 question marks) then *exactly one question mark* **MUST** also be removed. This preserves  
2703 the query component of the original QXRI if it was null or contained only question marks.

2704 **IMPORTANT:** Construction of HTTP(S) URIs for authority resolution service endpoints is defined  
2705 in section 9.1.10. Note that this involves an additional step taken after all URI construction steps  
2706 specified in this section are complete. In other words, if the URI element of an authority resolution  
2707 service endpoint includes an `append` attribute, the Next Authority Resolution Service URI **MUST**  
2708 be fully constructed according to the algorithm in this section before appending the Next Authority  
2709 String as defined in section 9.1.10.

2710 **WARNING:** Use of any value of the `append` attribute other than `authority` on the URI element  
2711 for an authority resolution service endpoint is **NOT RECOMMENDED** due to the complexity it  
2712 introduces.

### 2713 **13.6.2 The uric Parameter**

2714 The `uric` subparameter of the Resolution Output Format is used to govern whether a resolver  
2715 should perform construction of the URI automatically on behalf of a consuming application.  
2716 Following are the processing rules for this parameter:

- 2717 1. If `uric=true`, a resolver **MUST** apply the URI construction rules specified in section  
2718 13.6.1 to each `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD in the  
2719 resolution chain. Note that this step is identical to the processing a resolver must perform  
2720 to output a URI list.
- 2721 2. The resolver **MUST** replace the value of each `xrd:XRD/xrd:Service/xrd:URI`  
2722 element in the final XRD with the fully constructed URI value.
- 2723 3. The resolver **MUST** subsequently remove the `append` attribute from each  
2724 `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD.
- 2725 4. If `uric=false` or the parameter is absent or empty, a resolver **MUST NOT** perform any  
2726 of the processing specified in this section.

2727

## 14 Synonym Verification

2728 As described in section 5, a consuming application must be able to verify the security of the  
2729 binding between the fully-qualified query identifier (the identifier resolved to an XRDS document)  
2730 and any synonyms asserted in the final XRD. This section defines synonym verification rules.

### 14.1 Redirect Verification

2732 As specified in section 12.3, XRI resolvers MUST verify the synonyms asserted in the XRD  
2733 obtained by following a Redirect element. These rules are:

- 2734 1. If resolution of the Redirect succeeds, the resolver MUST first verify that the set of XRD  
2735 synonym elements (as specified in section 5.2) contained in the new XRD are *equivalent*  
2736 *to or a subset of* those contained in the XRD containing the Redirect.
- 2737 2. Secondly, the resolver MUST verify that the content of each synonym element contained  
2738 in the new XRD is exactly equivalent to the content of the corresponding element in the  
2739 XRD containing the Redirect.
- 2740 3. If either rule above fails, the resolver MUST stop and return the error 253  
2741 REDIRECT\_VERIFY\_FAILED in the XRD where the error occurred or as a plain text error  
2742 message as defined in section 15.

2743 For examples see section 12.5.1.

### 14.2 EquivID Verification

2745 Although XRI resolvers do not automatically perform EquivID synonym verification, a consuming  
2746 application can easily request it using the following steps:

- 2747 1. First request resolution for the original query identifier with CanonicalID verification  
2748 enabled (`cid=true`).
- 2749 2. From the final XRD in the resolution chain, select the EquivID for which verification is  
2750 desired.
- 2751 3. Request resolution of the EquivID identifier.
- 2752 4. From the final XRD in this second resolution chain, determine if there is either: a) a  
2753 `xrd:XRD/xrd:EquivID` element, or b) a `xrd:XRD/xrd:CanonicalEquivID` element  
2754 whose value matches the verified CanonicalID of the original query identifier. If there is a  
2755 match, the EquivID is verified; otherwise it is not verified.

#### Example:

- 2757 • Fully-Qualified Query Identifier: `http://example.com/user`
- 2758 • Asserted EquivID: `xri://=!1000.c78d.402a.8824.bf20`

2759 First XRDS (for `http://example.com/user` — simplified for illustration purposes):

```
2760 <XRDS>  
2761 <XRD>  
2762 <EquivID>xri://=!1000.c78d.402a.8824.bf20</EquivID>  
2763 <CanonicalID>http://example.com/user</CanonicalID>  
2764 <Service priority="10">  
2765 ...  
2766 </Service>  
2767 ...  
2768 </XRD>  
2769 </XRDS>
```

2770 Second XRDS (for `xri://=!1000.c78d.402a.8824.bf20`):

```
2771 <XRDS>
2772 <XRD>
2773 <Query>!1000.c78d.402a.8824.bf20</Query>
2774 <ProviderID>xri://=</ProviderID>
2775 <EquivID>http://example.com/user</EquivID>
2776 <CanonicalID>xri://=!1000.c78d.402a.8824.bf20</CanonicalID>
2777 <Service priority="10">
2778   ...
2779 </Service>
2780   ...
2781 </XRD>
2782 </XRDS>
```

2783 The EquivID is verified because the XRD in the second XRDS asserts an EquivID backpointer to  
2784 the CanonicalID of the XRD in the first XRDS.

## 2785 14.3 CanonicalID Verification

2786 XRI resolvers automatically perform verification of CanonicalID and CanonicalEquivID synonyms  
2787 unless this function is explicitly turned off using the Resolution Output Format subparameter `cid`.  
2788 The following synonym verification MUST be applied by an XRI resolver if `cid=true` or the  
2789 parameter is absent or empty, and MUST NOT be applied if `cid=false`.

- 2790 1. If the value of the `xrd:XRD/xrd:CanonicalID` element is an HTTP(S) URI, it MUST  
2791 be verified as specified in section 14.3.1.
- 2792 2. If the value of the `xrd:XRD/xrd:CanonicalID` element is an XRI, it MUST be verified  
2793 as specified in section 14.3.2.
- 2794 3. If the value of the `xrd:XRD/xrd:CanonicalID` element is any other identifier,  
2795 CanonicalID verification fails and the resolver MUST return the CanonicalID verification  
2796 status specified in section 14.3.4.
- 2797 4. If CanonicalID verification succeeds but the final XRD in the resolution chain also  
2798 contains a `xrd:XRD/xrd:CanonicalEquivID` element, it MUST also be verified as  
2799 specified in section 14.3.3, and the resolver MUST return the CanonicalEquivID  
2800 verification status as specified in section 14.3.4.
- 2801 5. In all cases, since synonym verification depends on trusting each authority in the  
2802 resolution chain, trusted resolution (section 10) SHOULD be used with either  
2803 `https=true` or `saml=true` or both to provide additional assurance of the authenticity of  
2804 the results.

2805 **IMPORTANT:** There is no guarantee that all XRDS that describe the same target resource will  
2806 return the same verified CanonicalID or CanonicalEquivID. Different parent authorities may assert  
2807 different CanonicalIDs or CanonicalEquivIDs for the same target resource and all of these may all  
2808 be verifiable. In addition, due to Redirect and Ref processing, the verified CanonicalID or  
2809 CanonicalEquivID returned for an XRI MAY differ depending on the resolution input parameters.  
2810 For example, as described in section 12, a request for a specific service endpoint type may  
2811 trigger processing of a Redirect or Ref resulting in a nested XRDS document. The final XRD in  
2812 the nested XRDS document may come from a different parent authority and have a different but  
2813 still verifiable CanonicalID or CanonicalEquivID.

### 2814 14.3.1 HTTP(S) URI Verification Rules

2815 To verify that an HTTP(S) URI is a valid CanonicalID synonym for a fully-qualified query identifier  
2816 (defined in section 5.1), a resolver MUST verify that all the following tests are successful:

- 2817 1. The fully-qualified query identifier MUST also be an HTTP(S) URI.

- 2818 2. The query identifier MUST be resolved as specified in section 6.
- 2819 3. The asserted CanonicalID synonym MUST be an HTTP(S) URI equivalent to: a) the fully-
- 2820 qualified query identifier, or b) the fully-qualified query identifier plus a valid fragment as
- 2821 defined by [RFC3986].

2822 See the example in section 14.3.5.

### 2823 14.3.2 XRI Verification Rules

2824 To verify that an XRI is a valid CanonicalID synonym for a fully-qualified query identifier (defined

2825 in section 5.1), a resolver MUST verify that all the following tests are successful.

- 2826 1. In the first XRD in the resolution chain for the fully-qualified query identifier, the value of
- 2827 the `xrd:XRD/xrd:ProviderID` element in the XRD from the community root authority
- 2828 MUST match the value of the `xrd:XRD/xrd:CanonicalID` element configured in the
- 2829 XRI resolver or available in a self-describing XRD from the community root authority (or
- 2830 its equivalent). See section 9.1.6.
- 2831 2. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID`
- 2832 element MUST be a direct child authority of the value of the
- 2833 `xrd:XRD/xrd:ProviderID` element. i.e., the former MUST consist of the latter plus
- 2834 one additional XRI subsegment as defined in [XRISyntax]. For example, if the value of
- 2835 the `xrd:XRD/xrd:CanonicalID` element is `@!1`, then the the value of the
- 2836 `xrd:XRD/xrd:ProviderID` element must be `@`.
- 2837 3. For each subsequent XRD in the resolution chain, the value of the
- 2838 `xrd:XRD/xrd:CanonicalID` element MUST be a direct child authority of the value of
- 2839 the `xrd:XRD/xrd:CanonicalID` element of the preceding XRD in the same XRDS
- 2840 document. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element
- 2841 asserted in an XRD is `@!1!2!3`, then the value of the `xrd:XRD/xrd:CanonicalID`
- 2842 element in the XRD of the parent authority must be `@!1!2`.
- 2843 4. If Redirect or Ref processing is required during resolution as specified in section 12, the
- 2844 rules above MUST also apply for each nested XRDS document.

Comment [DSR12]: PROOF – Added per John's suggestion to clarify how CanonicalID verification chains work.

2845 IMPORTANT: each set of XRDs in each new nested XRDS document produced as a result of

2846 Redirect or Ref processing constitutes its own CanonicalID verification chain. *CanonicalID*

2847 *verification never crosses between XRDS documents.* See the examples in section 12.5.

### 2848 14.3.3 CanonicalEquivID Verification

2849 CanonicalID verification also requires verification of a CanonicalEquivID *only if it is present in the*

2850 *final XRD in the resolution chain.* Since CanonicalEquivID verification typically requires an extra

2851 resolution cycle, restricting automatic verification to the final XRD in the resolution chain ensures

2852 it will add at most one additional resolution cycle.

2853 CanonicalEquivID verification MUST NOT be performed unless CanonicalID verification as

2854 specified in section 14.3 has completed successfully. The resulting value is called the *verified*

2855 *CanonicalID*.

2856 To verify that a CanonicalEquivID is an authorized synonym for a verified CanonicalEquivID, a

2857 resolver MUST either verify that the value of the CanonicalEquivID element is character-by-

2858 character equivalent to the verified CanonicalID (since both appear in the same XRD, all other

2859 normalization rules are waived), or all the following tests MUST be successful:

- 2860 1. The asserted CanonicalEquivID value MUST be a valid HTTP(S) URI or XRI.
- 2861 2. The asserted CanonicalEquivID value MUST resolve successfully to an XRDS document
- 2862 according to the rules in this specification *using the same resolution parameters as in the*
- 2863 *original resolution request.*

Comment [DSR13]: PROOF: This text was added per input from John and Steve that the exception added in ED08 required that the CanonicalEquivID and CanonicalID be equivalent. This is a much more direct way to state it.

- 2864 3. The CanonicalID in the final XRD of the resolved XRDS document MUST be equivalent  
2865 to the asserted CanonicalEquivID.
- 2866 4. The final XRD in the resolved XRDS document MUST contain either an EquivID or a  
2867 CanonicalEquivID "backpointer" whose value is equivalent to the verified CanonicalID.

**Comment [DSR14]:** PROOF – Added per suggestion from John and Steve that a CanonicalEquivID should always match the CanonicalID of its target XRD once it is resolved.

2868 **SPECIAL SECURITY CONSIDERATION:** See section 5.2.2 regarding the rules for provisioning  
2869 of `xrd:XRD/xrd:EquivID` and `xrd:XRD/xrd:CanonicalEquivID` elements in an XRD.

### 2870 14.3.4 Verification Status Attributes

2871 If CanonicalID verification is performed, an XRI resolver MUST return the CanonicalID and  
2872 CanonicalEquivID verification status using an attribute of the `xrd:XRD/xrd:Status` element in  
2873 each XRD in the output as follows:

- 2874 1. CanonicalID verification MUST be reported using the `cid` attribute.
- 2875 2. CanonicalEquivID verification MUST be reported using the `ceid` attribute.
- 2876 3. Both attributes accept four enumerated values: `absent` if the element is not present, `off`  
2877 if verification is not performed, `verified` if the element is verified, and `failed` if  
2878 verification fails.
- 2879 4. The `off` value applies to both elements if CanonicalID verification is not performed  
2880 (`cid=false`).
- 2881 5. The `off` value applies to the CanonicalEquivID element in any XRD before the final XRD  
2882 if CanonicalID verification is performed (`cid=true`), because a resolver only verifies this  
2883 element in the final XRD.
- 2884 6. If `cid=true` and verification of any CanonicalID element fails, *verification of all*  
2885 *CanonicalIDs in all subsequent XRDs in the same XRDS document MUST fail.*

**Comment [DSR15]:** PROOF – Added to clarify that failure of one CanonicalID causes all subsequent XRDs in that XRDS document to fail CanonicalID verification.

2886 From these verification status attributes, a consuming application can confirm on every XRD in  
2887 the XRDS document whether the CanonicalID is present and has been verified. In addition, for  
2888 the final XRD in the XRDS document, it can confirm whether the CanonicalEquivID element is  
2889 present and has been verified.

### 2890 14.3.5 Examples

#### 2891 Example #1:

- 2892 • Fully-Qualified Query Identifier: `http://example.com/user`
- 2893 • Asserted CanonicalID: `http://example.com/user#1234`

2894 XRDS (simplified for illustration purposes):

```
2895 <XRDS ref="http://example.com/user">  
2896 <XRD>  
2897 <CanonicalID>http://example.com/user#1234</CanonicalID>  
2898 <Service priority="10">  
2899 ...  
2900 </Service>  
2901 ...  
2902 </XRD>  
2903 </XRDS>
```

2904 The asserted CanonicalID satisfies the HTTP(S) URI verification rules in section 14.3.1.

2905

2906 **Example #2:**

- 2907 • Fully-Qualified Query Identifier: =example.name\*delegate.name  
2908 • Asserted CanonicalID: !=1000.62b1.44fd.2855!1234

2909 XRDS (for =example.name\*delegate.name):

```
2910 <XRDS ref="xri://=example.name*delegate.name">
2911   <XRD>
2912     <Query>*example.name</Query>
2913     <ProviderID>xri://=</ProviderID>
2914     <LocalID>!1000.62b1.44fd.2855</LocalID>
2915     <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2916     <Service>
2917       <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2918       <Type>xri://$res*auth*($v*2.0)</Type>
2919       <MediaType>application/xrds+xml</MediaType>
2920       <URI priority="10">http://resolve.example.com</URI>
2921       <URI priority="15">http://resolve2.example.com</URI>
2922       <URI>https://resolve.example.com</URI>
2923     </Service>
2924     ...
2925   </XRD>
2926   <XRD>
2927     <Query>*delegate.name</Query>
2928     <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
2929     <LocalID>!1234</LocalID>
2930     <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
2931     <Service priority="1">
2932       ...
2933     </Service>
2934     ...
2935   </XRD>
2936 </XRDS>
```

2937 The asserted CanonicalID satisfies the XRI verification rules in section 14.3.2.

2938

---

2939 **Example #3:**

- 2940 • Fully-Qualified Query Identifier: http://example.com/user  
2941 • Asserted CanonicalID: http://example.com/user  
2942 • Asserted CanonicalEquivID: https://different.example.net/path/user

2943 First XRDS (for http://example.com/user):

```
2944 <XRDS ref="http://example.com/user">
2945   <XRD>
2946     <CanonicalID>http://example.com/user</CanonicalID>
2947     <CanonicalEquivID>
2948       https://different.example.net/path/user
2949     </CanonicalEquivID>
2950     <Service priority="10">
2951       ...
2952     </Service>
2953     ...
2954   </XRD>
2955 </XRDS>
```

2956 Second XRDS (for https://different.example.net/path/user):

```

2957 <XRDS ref="https://different.example.net/path/user">
2958 <XRD>
2959 <EquivID>http://example.com/user</EquivID>
2960 <CanonicalID>https://different.example.net/path/user</CanonicalID>
2961 <Service priority="10">
2962 ...
2963 </Service>
2964 ...
2965 </XRD>
2966 </XRDS>

```

2967 The CanonicalEquivID asserted in the first XRDS satisfies the verification rules in section 14.3.3  
 2968 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of  
 2969 the first XRDS.

2970

---

2971 **Example #4:**

- 2972 • Fully-Qualified Query Identifier: http://example.com/user
- 2973 • Asserted CanonicalID: http://example.com/user
- 2974 • Asserted CanonicalEquivID: =!1000.62b1.44fd.2855

2975 XRDS (for http://example.com/user):

```

2976 <XRDS ref="http://example.com/user">
2977 <XRD>
2978 <CanonicalID>http://example.com/user</CanonicalID>
2979 <CanonicalEquivID>xri://=!1000.62b1.44fd.2855</CanonicalEquivID>
2980 <Service priority="10">
2981 ...
2982 </Service>
2983 ...
2984 </XRD>
2985 </XRDS>

```

2986 XRDS (for xri://=!1000.62b1.44fd.2855):

```

2987 <XRDS ref="xri://=!1000.62b1.44fd.2855">
2988 <XRD>
2989 <Query>!1000.62b1.44fd.2855</Query>
2990 <ProviderID>xri://=!</ProviderID>
2991 <EquivID>http://example.com/user</EquivID>
2992 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
2993 <Service priority="10">
2994 ...
2995 </Service>
2996 ...
2997 </XRD>
2998 </XRDS>

```

2999 The CanonicalEquivID asserted in the first XRDS satisfies the verification rules in section 14.3.3  
 3000 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of  
 3001 the first XRDS.

3002

---

3003 **Example #5:**

- 3004 • Fully-Qualified Query Identifier: =example.name
- 3005 • Asserted CanonicalID: xri://=!1000.62b1.44fd.2855

- 3006 • Asserted CanonicalEquivID: `https://example.com/user`

3007 First XRDS (for `=example.name`):

```
3008 <XRDS ref="xri://=example.name">
3009 <XRD>
3010 <Query>*example.name</Query>
3011 <ProviderID>xri://= </ProviderID>
3012 <LocalID>!1000.62b1.44fd.2855</LocalID>
3013 <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
3014 <CanonicalEquivID>https://example.com/user</CanonicalEquivID>
3015 <Service priority="10">
3016 ...
3017 </Service>
3018 ...
3019 </XRD>
3020 </XRDS>
```

3021 Second XRDS (for `https://example.com/user`):

```
3022 <XRDS ref="https://example.com/user">
3023 <XRD>
3024 <EquivID>xri://=!1000.62b1.44fd.2855</EquivID>
3025 <CanonicalID>https://example.com/user</CanonicalID>
3026 <Service priority="10">
3027 ...
3028 </Service>
3029 ...
3030 </XRD>
3031 </XRDS>
```

3032 The CanonicalEquivID asserted in the first XRDS satisfies the verification rules in section 14.3.3  
3033 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of  
3034 the first XRDS.

3035

---

### 3036 Example #6:

- 3037 • Fully-Qualified Query Identifier: `=example.name*delegate.name`
- 3038 • Asserted CanonicalID: `xri://=!1000.62b1.44fd.2855!1234`
- 3039 • Asserted CanonicalEquivID: `@!1000.f3da.9056.aca3!5555`

3040 First XRDS (for `=example.name*delegate.name`):

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

```

3058 <Query>*delegate.name</Query>
3059 <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
3060 <LocalID>!1234</LocalID>
3061 <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
3062 <CanonicalEquivID>
3063   xri://@11000.f3da.9056.aca3!5555
3064 </CanonicalEquivID>
3065 <Service priority="1">
3066   ...
3067 </Service>
3068   ...
3069 </XRD>
3070 </XRDS>

```

3071 • Second XRDS (for @!1000.f3da.9056.aca3!5555):

```

3072 <XRDS ref="xri://@!1000.f3da.9056.aca3!5555">
3073 <XRD>
3074 <Query>!1000.f3da.9056.aca3</Query>
3075 <ProviderID>xri://@</ProviderID>
3076 <CanonicalID>xri://@!1000.f3da.9056.aca3</CanonicalID>
3077 <Service>
3078 <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
3079 <Type>xri://$res*auth*($v*2.0)</Type>
3080 <MediaType>application/xrds+xml</MediaType>
3081 <URI priority="10">http://resolve.example.com</URI>
3082 <URI priority="15">http://resolve2.example.com</URI>
3083 <URI>https://resolve.example.com</URI>
3084 </Service>
3085   ...
3086 </XRD>
3087 <XRD>
3088 <Query>!5555</Query>
3089 <ProviderID>xri://@!1000.f3da.9056.aca3</ProviderID>
3090 <LocalID>!5555</LocalID>
3091 <EquivID>xri://=!1000.62b1.44fd.2855!1234</EquivID>
3092 <CanonicalID>xri://@!1000.f3da.9056.aca3!5555</CanonicalID>
3093 <Service priority="1">
3094   ...
3095 </Service>
3096   ...
3097 </XRD>
3098 </XRDS>

```

3099 The CanonicalEquivID asserted in the final XRD of the first XRDS satisfies the verification rules  
3100 in section 14.3.3 because it resolves to a second XRDS whose final XRD asserts an EquivID  
3101 backpointer to the CanonicalID of the final XRD in the first XRDS.

---

## 3102 15 Status Codes and Error Processing

### 3103 15.1 Status Elements

3104 XRDS architecture uses two XRD elements for status reporting:

- 3105 • The `xrd:XRD/xrd:ServerStatus` element is used by an authority server to report the  
3106 server-side status of a resolution query to a resolver.
- 3107 • The `xrd:XRD/xrd:Status` element is used by a resolver to report the client-side status of  
3108 a resolution query to a consuming application. Note that attributes and contents of this  
3109 element MAY differ from those of the `xrd:XRD/xrd:ServerStatus` element due to either  
3110 client-side error detection or reporting of CanonicalID verification status (section 14.3.4).

3111 Following are the normative rules that apply to usage of these elements:

- 3112 1. For XRDS servers and clients, each of these elements is OPTIONAL.
- 3113 2. An XRI authority server is REQUIRED to include an `xrd:XRD/xrd:ServerStatus`  
3114 element for each XRD in a resolution response.

3115 **BACKWARDS COMPATABILITY NOTE:** The `xrd:XRD/xrd:ServerStatus` element was not  
3116 included in earlier versions of this specification. If an older authority resolution server does not  
3117 produce this element in generic or HTTPS trusted resolution, a resolver SHOULD generate it. For  
3118 SAML trusted resolution, a resolver MUST NOT generate it.

- 3119 3. An XRI resolver is REQUIRED to add an `xrd:XRD/xrd:Status` element to each XRD  
3120 If the Resolution Output Format is an XRDS document or an XRD element.
- 3121 4. In SAML trusted resolution, a resolver MUST verify the SAML signature on the XRD  
3122 received from the server as specified in section 10.2.4 before adding the  
3123 `xrd:XRD/xrd:Status` element to the XRD. Because this modifies the XRD, a  
3124 consuming application may not be able to easily verify the SAML signature itself. Should  
3125 this be necessary, the consuming application may request the XRD it wishes to verify  
3126 directly from an authority server using the SAML trusted resolution protocol in section  
3127 10.2.
- 3128 5. These elements MUST include the status codes specified in section 15.2 as the value of  
3129 the required `code` attribute.
- 3130 6. These elements SHOULD contain the status context strings specified in section 15.3.  
3131 Authority servers or resolvers MAY add additional information to status context strings.

### 3132 15.2 Status Codes

3133 XRI resolution status codes are patterned after the HTTP model. They are broken into three  
3134 major categories:

- 3135 • 1xx: Success—the requested resolution operation was completed successfully.
- 3136 • 2xx: Permanent errors—the resolver encountered an error from which it could not recover.
- 3137 • 3xx: Temporary errors—the resolver encountered an error condition that may be only  
3138 temporary.

3139 The 2xx and 3xx categories are broken into seven minor categories:

- 3140 • x0x: General error that may take place during any phase of resolution.

- 3141 • x1x: Input error
- 3142 • x2x: Generic authority resolution error.
- 3143 • x3x: Trusted authority resolution error.
- 3144 • x4x: Service endpoint (SEP) selection error.
- 3145 • x5x: Redirect error.
- 3146 • x6x: Ref error.

3147 The full list of XRI resolution status codes is defined in Table 29.

3148

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 <code>xrd:Query</code> element does not match the subsegment requested.
224	INACTIVE	Authority resolution	The query XRI has been assigned but the authority does not provide resolution metadata.
230	TRUSTED_RES_ERROR	Trusted	Generic trusted resolution error.

		resolution	
231	HTTPS_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS authority resolution endpoint.
232	SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate a SAML authority resolution endpoint.
233	HTTPS+SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS+SAML authority resolution endpoint.
234	UNVERIFIED_SIGNATURE	Trusted resolution	Signature verification failed.
240	SEP_SELECTION_ERROR	SEP selection	Generic service endpoint selection error.
241	SEP_NOT_FOUND	SEP selection	The requested service endpoint could not be found in the current XRD or via Redirect or Ref processing.
250	REDIRECT_ERROR	Redirect Processing	Generic Redirect error.
251	INVALID_REDIRECT	Redirect Processing	At least one Redirect element was found but resolution failed.
252	INVALID_HTTPS_REDIRECT	Redirect Processing	<code>https=true</code> but a Redirect element containing an HTTPS URI was not found.
253	REDIRECT_VERIFY_FAILED	Redirect Processing	Synonym verification failed in an XRD after following a redirect. See section 12.3
260	REF_ERROR	Ref Processing	Generic Ref processing error.
261	INVALID_REF	Ref Processing	A valid Ref XRI was not found.
262	REF_NOT_FOLLOWED	Ref Processing	At least one Ref was present but the <code>refs</code> parameter was set to <code>false</code> .
300	TEMPORARY_FAIL	Any	Generic temporary failure.
301	TIMEOUT_ERROR	Any	Locally-defined timeout limit has lapsed during an operation (e.g. network latency).
320	NETWORK_ERROR	Authority resolution	Generic error during authority resolution phase (includes uncaught exception, system error, network error).
321	UNEXPECTED_RESPONSE	Authority resolution	When querying an authority server, the server returned a non-200 HTTP status.
322	INVALID_XRDS	Authority resolution	Invalid XRDS received from an authority server (includes malformed XML, truncated content, or wrong content)

			type).
--	--	--	--------

3149 *Table 29: Error codes for XRI resolution.*

### 3150 **15.3 Status Context Strings**

3151 Each status code in Table 29 MAY be returned with an optional status context string that provides  
3152 additional human-readable information about the status or error condition. When the Resolution  
3153 Output Format is an XRDS document or XRD element, this string is returned as the contents of  
3154 the `xrd:XRD/xrd:ServerStatus` and `xrd:XRD/xrd:Status` elements. When the  
3155 Resolution Output Format is a URI List, this string MUST be returned as specified in section 15.4.  
3156 Implementers SHOULD provide error context strings with additional information about an error  
3157 and possible solutions whenever it can be helpful to developers or end users.

### 3158 **15.4 Returning Errors in Plain Text or HTML**

3159 If the Resolution Output Format is a URI List as defined in section 8.2, an error MUST be  
3160 returned with the content type `text/plain`. In this content:

- 3161 • The first line MUST consist of only the numeric error code as defined in section 15.2 followed  
3162 by a CRLF.
- 3163 • The second line is RECOMMENDED; if present it MUST contain the error context string as  
3164 defined in section 15.3.

3165 The same rules apply if the Resolution Output Format is an HTTP(S) Redirect as defined in  
3166 section 8.2, except the media type MAY also be `text/html`. It is particularly important in this  
3167 case to return an error message that will be understandable to an end-user who may have no  
3168 knowledge of XRI resolution or the fact that the error is coming from an XRI proxy resolver.

### 3169 **15.5 Error Handling in Recursing and Proxy Resolution**

3170 In recursing and proxy resolution (sections 9.1.8 and 11), a server is acting as a client resolver for  
3171 other authority resolution service endpoints. If in this intermediary capacity it receives an  
3172 unrecoverable error, it MUST return the error to the originating client in the output format  
3173 specified by the value of the requested Resolution Output Format as defined in section 8.2.

3174 If the output format is an XRDS document, it MUST contain `xrd:XRD` elements for all  
3175 subsegments successfully resolved or retrieved from cache prior to the error. Each XRD MUST  
3176 include the `xrd:ServerStatus` element as reported by the authoritative server. The final  
3177 `xrd:XRD` element MUST include the `xrd:Query` element that produced the error and the  
3178 `xrd:Status` element that describes the error as defined above.

3179 If the output format is an XRD element, it MUST include the `xrd:Query` element that produced  
3180 the error, the `xrd:ServerStatus` element as reported by the authoritative server, and the  
3181 `xrd:Status` element that describes the error as defined above.

3182 If this output format is a URI List or an HTTP(S) redirect, a proxy resolver SHOULD return a  
3183 human-readable error message as specified in section 15.4.

---

## 3184 16 Use of HTTP(S)

### 3185 16.1 HTTP Errors

3186 When a resolver encounters fatal HTTP(S) errors during the resolution process, it **MUST** return  
3187 the appropriate XRI resolution error code and error message as defined in section 15. In this way  
3188 calling applications do not have to deal separately with XRI and HTTP error messages.

### 3189 16.2 HTTP Headers

#### 3190 16.2.1 Caching

3191 The HTTP caching capabilities described by **[RFC2616]** should be leveraged for all XRDS and  
3192 XRI resolution protocols. Specifically, implementations **SHOULD** implement the caching model  
3193 described in section 13 of **[RFC2616]**, and in particular, the “Expiration Model” of section 13.2, as  
3194 this requires the fewest round-trip network connections.

3195 All XRI resolution servers **SHOULD** send the Cache-Control or Expires headers in their  
3196 responses per section 13.2 of **[RFC2616]** unless there are overriding security or policy reasons to  
3197 omit them.

3198 Note that HTTP Cache headers **SHOULD NOT** conflict with expiration information in an XRD.  
3199 That is, the expiration date specified by HTTP caching headers **SHOULD NOT** be later than any  
3200 of the expiration dates for any of the `xrd:Expires` elements returned in the HTTP response.  
3201 This implies that recursing and proxy resolvers **SHOULD** compute the “soonest” expiration date  
3202 for the XRDs in a resolution chain and ensure a later date is not specified by the HTTP caching  
3203 headers for the HTTP response.

#### 3204 16.2.2 Location

3205 During HTTP interaction, “Location” headers may be present per **[RFC2616]** (i.e., during 3XX  
3206 redirects). Redirects **SHOULD** be made cacheable through appropriate HTTP headers, as  
3207 specified in section 16.2.1.

#### 3208 16.2.3 Content-Type

3209 For authority resolution, the Content-Type header in the 2XX responses **MUST** contain the media  
3210 type identifier values specified in Table 11 (for generic resolution), Table 15 (for HTTPS trusted  
3211 resolution), Table 16 (for SAML trusted resolution), or Table 17 (or HTTPS+SAML trusted  
3212 resolution).

3213 Following the optional service endpoint selection phase, clients and servers **MAY** negotiate  
3214 content type using standard HTTP content negotiation features. Regardless of whether this  
3215 feature is used, however, the server **MUST** respond with an appropriate media type in the  
3216 Content-Type header if the resource is found and an appropriate content type is returned.

### 3217 16.3 Other HTTP Features

3218 HTTP provides a number of other features including transfer-coding, proxying, validation-model  
3219 caching, and so forth. All these features may be used insofar as they do not conflict with the  
3220 required uses of HTTP described in this document.

## 3221 16.4 Caching and Efficiency

### 3222 16.4.1 Resolver Caching

3223 In addition to HTTP-level caching, resolution clients are encouraged to perform caching at the  
3224 application level. For best results, however, resolution clients SHOULD be conservative with  
3225 caching expiration semantics, including cache expiration dates. This implies that in a series of  
3226 HTTP redirects, for example, the results of the entire process SHOULD only be cached as long  
3227 as the shortest period of time allowed by any of the intermediate HTTP responses.

3228 Because not all HTTP client libraries expose caching expiration to applications, identifier  
3229 authorities SHOULD NOT use cacheable redirects with expiration times sooner than the  
3230 expiration times of other HTTP responses in the resolution chain. In general, all XRI deployments  
3231 should be mindful of limitations in current HTTP clients and proxies.

3232 The cache expiration time of an XRD may also be explicitly limited by the parent authority. If the  
3233 expiration time in the `xrd:Expires` element is sooner than the expiration time calculated from  
3234 the HTTP caching semantics, the XRD MUST be discarded before the expiration time in  
3235 `xrd:Expires`. Note also that a `saml:Assertion` element returned during SAML trusted  
3236 resolution has its own signature expiration semantics as defined in [SAML]. While this may  
3237 invalidate the SAML signature, a resolver MAY still use the balance of the contents of the XRD if  
3238 it is not expired by HTTP caching semantics or the `xrd:Expires` element.

3239 With both application-level and HTTP-level caching, the resolution process is designed to have  
3240 minimal overhead. Resolution of each qualified subsegment of an XRI authority component is a  
3241 separate step described by a separate XRD, so intermediate results can typically be cached in  
3242 their entirety. For this reason, resolution of higher-level (i.e., further to the left) qualified  
3243 subsegments, which are common to more identifiers, will naturally result in a greater number of  
3244 cache hits than resolution of lower-level subsegments.

### 3245 16.4.2 Synonyms

3246 The publication of synonyms in XRDS documents (section 5) can further increase cache  
3247 efficiency. If an XRI resolution request produces a cache hit on a synonym, the following rules  
3248 apply:

- 3249 1. If the cache hit is on a LocalID synonym, the resolver MAY return the cached XRD  
3250 element if: a) it is from the correct ProviderID, b) it has not expired, and c) it was obtained  
3251 using the same trusted resolution and synonym verification parameters as the current  
3252 resolution request.
- 3253 2. If the cache hit is on a CanonicalID synonym, the resolver MAY return the entire cached  
3254 XRDS document if: a) it has not expired, and b) it was obtained using the same trusted  
3255 resolution and synonym verification parameters as the current resolution request.

3256 **IMPORTANT:** The effect of these rules is that the application calling an XRI resolver MAY receive  
3257 back an XRD element, or an XRDS document containing XRD element(s), in which the value of  
3258 the `<xrd:Query>` element does not match the resolution request, but in which the value of an  
3259 `<xrd:LocalID>` element does match the resolution request. This is acceptable for the generic  
3260 and HTTPS trusted resolution protocols but not the SAML trusted resolution protocol, where the  
3261 value of the `<xrd:Query>` element MUST match the resolution request as specified in section  
3262 10.2.4.

3263

## 17 Extensibility and Versioning

3264

### 17.1 Extensibility

3265

#### 17.1.1 Extensibility of XRDs

3266

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.

3273

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.

3274

3275

3276

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.

3277

3278

3279

3280

3281

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`):

3282

3283

3284

3285

3286

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

3287

3288

3289

3290

3291

3292

3293

3294

3295

3296

3297

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.

3298

3299

3300

3301

3302

3303

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

3304

3305

#### 17.1.2 Other Points of Extensibility

3306

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:

3307

- 3308 • Specification of new resolution service types or other service types using XRI, IRIs, or URIs  
3309 as values of the `xrd:Type` element.
- 3310 • Specification of new resolution output formats or features using media types and media type  
3311 parameters as values of the `xrd:MediaType` element as defined in [RFC2045] and  
3312 [RFC2046].
- 3313 • HTTP negotiation of content types, language, encoding, etc. as defined by [RFC2616].
- 3314 • Use of HTTP redirects (3XX) or other response codes defined by [RFC2616].
- 3315 • Use of cross-references within XRI, particularly for associating new types of metadata with a  
3316 resource. See [XRISyntax] and [XRIMetadata].

## 3317 17.2 Versioning

3318 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,  
3319 this section describes versioning guidelines.

3320 In general, this specification follows the same versioning guidelines as established in section  
3321 4.2.1 of [SAML]:

3322 *In general, maintaining namespace stability while adding or changing the content of a*  
3323 *schema are competing goals. While certain design strategies can facilitate such changes,*  
3324 *it is complex to predict how older implementations will react to any given change, making*  
3325 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*  
3326 *minor revisions is reserved, in the interest of namespace stability. Except in special*  
3327 *circumstances (for example, to correct major deficiencies or to fix errors),*  
3328 *implementations should expect forward-compatible schema changes in minor revisions,*  
3329 *allowing new messages to validate against older schemas.*

3330 *Implementations SHOULD expect and be prepared to deal with new extensions and*  
3331 *message types in accordance with the processing rules laid out for those types. Minor*  
3332 *revisions MAY introduce new types that leverage the extension facilities described in [this*  
3333 *section]. Older implementations SHOULD reject such extensions gracefully when they*  
3334 *are encountered in contexts that dictate mandatory semantics.*

### 3335 17.2.1 Version Numbering

3336 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number  
3337 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version  
3338 number  $Major_A.Minor_A$  if and only if:

3339  $Major_B > Major_A$  OR (  $Major_B = Major_A$  ) AND  $Minor_B > Minor_A$  )

### 3340 17.2.2 Versioning of the XRI Resolution Specification

3341 New releases of the XRI Resolution specification may specify changes to the resolution protocols  
3342 and/or the XRD schema in Appendix B. When changes affect either of these, the resolution  
3343 service type version number will be changed. Where changes are purely editorial, the version  
3344 number will not be changed.

3345 In general, if a change is backward-compatible, the new version will be identified using the  
3346 current major version number and a new minor version number. If the change is not backward-  
3347 compatible, the new version will be identified with a new major version number.

### 3348 17.2.3 Versioning of Protocols

3349 The protocols defined in this document may also be versioned by future releases of the XRI  
3350 Resolution specification. If these protocols are not backward-compatible with older

3351 implementations, they will be assigned a new XRI with a new version identifier for use in  
3352 identifying their service type in XRDs. See section 3.1.2.

3353 Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP  
3354 provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an  
3355 XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely  
3356 to continue to use the same XRI to identify the protocol as was used in previous versions of the  
3357 XRI Resolution specification.

#### 3358 **17.2.4 Versioning of XRDs**

3359 The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have  
3360 no specific knowledge of the elements it may contain. Therefore it has no version indicator, and  
3361 can remain stable indefinitely because there is no need to version its namespace.

3362 The `xrd:XRD` element has a `version` attribute. This attribute is OPTIONAL for this version of  
3363 the XRI resolution specification (version 2.0). This attribute will be REQUIRED for all future  
3364 versions of this specification. When used, the value of this attribute MUST be the exact numeric  
3365 version value of the XRI Resolution specification to which its containing elements conform.

3366 When new versions of the XRI Resolution specification are released, the namespace for the XRD  
3367 schema may or may not be changed. If there is a major version number change, the namespace  
3368 for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the  
3369 namespace for the `xrd:XRD` schema may remain unchanged.

3370 Note that conformance to a specific XRD version does not preclude an author from including  
3371 extension elements from a different namespace in the XRD. See section 17.1 above.

3372

## 18 Security and Data Protection

3373  
3374  
3375  
3376

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.

3377

### 18.1 DNS Spoofing or Poisoning

3378  
3379  
3380  
3381  
3382  
3383  
3384  
3385  
3386

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.

3387

### 18.2 HTTP Security

3388  
3389  
3390  
3391

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.

3392  
3393  
3394  
3395

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.

3396  
3397

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

3398

### 18.3 SAML Considerations

3399  
3400  
3401

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

3402

### 18.4 Limitations of Trusted Resolution

3403  
3404  
3405  
3406  
3407

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.

3408  
3409  
3410

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.

## 3411 **18.5 Synonym Verification**

3412 As discussed in section 5, XRI and XRDS infrastructure has rich support for identifier synonyms,  
3413 including synonyms that cross security domains. For this reason it is particularly important that  
3414 identifier authorities, including registries, registrars, directory administrators, identity providers,  
3415 and other parties who issue XRIs and manage XRDS documents, enforce the security policies  
3416 highlighted in section 5 regarding registration and management of XRDS synonym elements.

## 3417 **18.6 Redirect and Ref Management**

3418 As discussed in sections 5.3 and 12, XRI and XRDS infrastructure includes the capability to  
3419 distribute and delegate XRDS document management across multiple network locations or  
3420 identifier authorities. Identifier authorities should follow the security precautions highlighted in  
3421 section 5.3 to ensure Redirects and Refs are properly authorized and represent the intended  
3422 delegation policies.

## 3423 **18.7 Community Root Authorities**

3424 The XRI authority information for a community root needs to be well-known to the clients that  
3425 request resolution within that community. For trusted resolution, this includes the authority  
3426 resolution service endpoint URIs, the `xrd:XRD/xrd:ProviderID`, and the `ds:KeyInfo`  
3427 information. An acceptable means of providing this information is for the community root authority  
3428 to produce a self-signed XRD and publish it to a server-authenticated HTTPS endpoint. Special  
3429 care should be taken to ensure the correctness of such an XRD; if this information is incorrect, an  
3430 attacker may be able to convince a client of an incorrect result during trusted resolution.

## 3431 **18.8 Caching Authorities**

3432 In addition to traditional HTTP caching proxies, XRI proxy resolvers may be a part of the  
3433 resolution topology. Such proxy resolvers should take special precautions against cache  
3434 poisoning, as these caching entities may represent trusted decision points within a deployment's  
3435 resolution architecture.

## 3436 **18.9 Recursing and Proxy Resolution**

3437 During recursing resolution, subsegments of the XRI authority component for which the resolving  
3438 network endpoint is not authoritative may be revealed to that service endpoint. During proxy  
3439 resolution, some or all of an XRI is provided to the proxy resolver.  
3440 In both cases, privacy considerations should be evaluated before disclosing such information.

## 3441 **18.10 Denial-Of-Service Attacks**

3442 XRI Resolution, including trusted resolution, is vulnerable to denial-of-service (DOS) attacks  
3443 typical of systems relying on DNS and HTTP(S).

3444

---

## A. Acknowledgments

3445 The editors would like to thank the following current and former members of the OASIS XRI TC  
3446 for their particular contributions to this and previous versions of this specification:

- 3447 • William Barnhill, Booz Allen and Hamilton
- 3448 • Dave McAlpin, Epok
- 3449 • Chetan Sabnis, Epok
- 3450 • Peter Davis, Neustar
- 3451 • Victor Grey, PlaNetwork
- 3452 • Mike Lindelsee, Visa International
- 3453 • Markus Sabadello, XDI.org
- 3454 • John Bradley
- 3455 • Kermit Snelson

3456 The editors would also like to acknowledge the contributions of the other members of the OASIS  
3457 XRI Technical Committee, whose other voting members at the time of publication were:

- 3458 • Geoffrey Strongin, Advanced Micro Devices
- 3459 • Ajay Madhok, AmSoft Systems
- 3460 • Dr. XiaoDong Lee, China Internet Network Information
- 3461 • Nat Sakimura, Nomura Research
- 3462 • Owen Davis, PlaNetwork
- 3463 • Fen Labalme, PlaNetwork
- 3464 • Marty Schleiff, The Boeing Company
- 3465 • Dave Wentker, Visa International
- 3466 • Paul Trevithick

3467 The editors also would like to acknowledge the following people for their contributions to previous  
3468 versions of OASIS XRI specifications (affiliations listed for OASIS members):

3469 Marc Le Maitre, Cordance Corporation; Thomas Bikeev, EAN International; Krishna Sankar,  
3470 Cisco; Winston Bumpus, Dell; Joseph Moeller, EDS; Steve Green, Epok; Lance Hood, Jerry  
3471 Kindall, Adarbad Master, Davis McPherson, Chetan Sabnis, and Loren West, Epok; Phillipe  
3472 LeBlanc, Jim Schreckengast, and Xavier Serret, Gemplus; John McGarvey, IBM; Reva Modi,  
3473 Infosys; Krishnan Rajagopalan, Novell; Masaki Nishitani, Tomonori Seki, and Tetsu Watanabe,  
3474 Nomura Research Institute; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling,  
3475 Verisign; Rajeev Maria, Terence Spielman, and John Veizades, Visa International; Lark Allen and  
3476 Michael Willett, Wave Systems; Matthew Dovey; Eamonn Neylon; Mary Nishikawa; Lars Marius  
3477 Garshol; Norman Paskin; and Bernard Vatant.

3478

## B. RelaxNG Schema for XRDS and XRD

3479 Following are links to the normative RelaxNG compact schema files for XRDS and XRD:

- 3480 • [TODO-CD – the final xrds.rnc file location will be listed here]
- 3481 • [TODO-CD – the final xrd.rnc file location will be listed here]

3482 **IMPORTANT:** The **xrd.rnc** schema does NOT include deprecated attribute values that are  
3483 recommended for backwards compatability. See the Backwards Compatability notes in the  
3484 specification for more details.

3485 Listings of these files are provided in this appendix for reference but are non-normative.

### 3486 **xrds.rnc**

```
3487 namespace xrds = "xri://$xrds"
3488 namespace xrd = "xri://$xrd*($v*2.0)"
3489 namespace local = ""
3490 datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
3491
3492 any.element =
3493   element * {
3494     (attribute * { text } *
3495     | text
3496     | any.element)*
3497   }
3498
3499 any.external.element =
3500   element * - (xrd:XRD | xrds:XRDS) {
3501     (attribute * { text } *
3502     | text
3503     | any.element)*
3504   }
3505
3506 other.attribute = attribute * - (local:*) {text}
3507
3508 start = XRDS
3509
3510 XRDS = element xrds:XRDS {
3511   other.attribute *,
3512   (attribute ref { xs:anyURI } | attribute redirect { xs:anyURI } )?,
3513   (any.external.element | XRDS | external "xrd.rnc" )*
3514 }
3515
```

### 3516 **xrd.rnc**

```
3517 default namespace = "xri://$xrd*($v*2.0)"
3518 namespace xrd = "xri://$xrd*($v*2.0)"
3519 namespace saml = "urn:oasis:names:tc:SAML:2.0:assertion"
3520 namespace ds = "http://www.w3.org/2000/09/xmldsig#"
3521 namespace local = ""
3522
3523 datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
3524
3525 start = XRD
3526
3527 anyelementbody =
3528   (attribute * {text}
3529   | text
3530   | element * {anyelementbody} )*
3531
3532 non.xrd.element = element * - xrd:* {
3533   anyelementbody
3534 }
3535
```

```

3536 other.attribute = attribute * - (local:* | xrd:* ) {text}
3537
3538
3539 XRD = element XRD {
3540   other.attribute *,
3541   attribute idref {xs:IDREF} ?,
3542   attribute version { "2.0" } ?,
3543   Query ?,
3544   Status ?,
3545   ServerStatus ?,
3546   Expires ?,
3547   ProviderID ?,
3548   (Redirect | Ref) ?,
3549   LocalID *,
3550   EquivID *,
3551   CanonicalID ?,
3552   CanonicalEquivID ?,
3553   Service *,
3554   element saml:Assertion {anyelementbody} ?,
3555   non.xrd.element *
3556 }
3557
3558 Query = element Query {
3559   other.attribute *,
3560   text
3561 }
3562
3563 statuspattern =
3564   other.attribute *,
3565   attribute code {xs:integer},
3566   attribute cid { "absent" | "off" | "verified" | "failed" } ?,
3567   attribute ceid { "absent" | "off" | "verified" | "failed" } ?,
3568   text
3569
3570 Status = element Status {
3571   statuspattern
3572 }
3573
3574 ServerStatus = element ServerStatus {
3575   statuspattern
3576 }
3577
3578 Expires = element Expires {
3579   other.attribute *,
3580   xs:dateTime
3581 }
3582
3583 ProviderID = element ProviderID {
3584   other.attribute *,
3585   xs:anyURI
3586 }
3587
3588 Redirect = element Redirect {
3589   other.attribute *,
3590   attribute priority {xs:integer}?,
3591   xs:anyURI
3592 }
3593
3594 Ref = element Ref{
3595   other.attribute *,
3596   attribute priority {xs:integer}?,
3597   xs:anyURI
3598 }
3599
3600 LocalID = element LocalID {
3601   other.attribute *,
3602   attribute priority {xs:integer} ?,
3603   xs:anyURI
3604 }
3605
3606 EquivID = element EquivID {

```

```

3607     other.attribute *,
3608     attribute priority {xs:integer} ?,
3609     xs:anyURI
3610 }
3611
3612 CanonicalID = element CanonicalID {
3613     other.attribute *,
3614     xs:anyURI
3615 }
3616
3617 CanonicalEquipID = element CanonicalEquipID {
3618     other.attribute *,
3619     xs:anyURI
3620 }
3621
3622 Service = element Service {
3623     other.attribute *,
3624     attribute priority {xs:integer}?,
3625     ProviderID?,
3626     Type *,
3627     Path *,
3628     MediaType *,
3629     (URI+|Redirect+|Ref+)?,
3630     LocalID *,
3631     element ds:KeyInfo {anyelementbody}?,
3632     non.xrd.element *
3633 }
3634
3635 URI = element URI {
3636     other.attribute *,
3637     attribute priority {xs:integer}?,
3638     attribute append {"none" | "local" | "authority" | "path" | "query" | "qxri"} ?,
3639     xs:anyURI
3640 }
3641
3642 selection.attributes = attribute match {"any" | "default" | "non-null" | "null" } ?,
3643     attribute select { xs:boolean } ?
3644
3645 Type = element Type {
3646     other.attribute *,
3647     selection.attributes,
3648     xs:anyURI
3649 }
3650
3651 Path = element Path {
3652     other.attribute *,
3653     selection.attributes,
3654     xs:string
3655 }
3656
3657 MediaType = element MediaType {
3658     other.attribute *,
3659     selection.attributes,
3660     xs:string
3661 }

```

3662

## C. XML Schema for XRDS and XRD

3663 Following are links to the non-normative W3C XML Schema files for XRDS and XRD. These are  
3664 provided for reference only as they are not able to fully express the extensibility semantics of the  
3665 RelaxNG versions.

- 3666 • [TODO-CD – the final xrds.xsd file location will be listed here]
- 3667 • [TODO-CD – the final xrd.xsd file location will be listed here]

3668 **IMPORTANT:** The **xrd.xsd** schema does NOT include deprecated attribute values that are  
3669 recommended for backwards compatability. See the Backwards Compatability notes in the  
3670 specification for more details.

3671 Listings of these files are provided in this appendix for reference.

### 3672 **xrds.xsd**

```
3673 <?xml version="1.0" encoding="UTF-8"?>
3674 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds"
3675 targetNamespace="xri://$xrds" elementFormDefault="qualified">
3676   <!-- Utility patterns -->
3677   <xs:attributeGroup name="otherattribute">
3678     <xs:anyAttribute namespace="##other" processContents="lax"/>
3679   </xs:attributeGroup>
3680   <xs:group name="otherelement">
3681     <xs:choice>
3682       <xs:any namespace="##other" processContents="lax"/>
3683       <xs:any namespace="##local" processContents="lax"/>
3684     </xs:choice>
3685   </xs:group>
3686   <!-- Patterns for elements -->
3687   <xs:element name="XRDS">
3688     <xs:complexType>
3689       <xs:sequence>
3690         <xs:group ref="xrds:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3691       </xs:sequence>
3692       <xs:attributeGroup ref="xrds:otherattribute"/>
3693       <!--XML Schema does not currently offer a means to express that only one of
3694 the following two attributes may be used in any XRDS element, i.e., an XRDS document may
3695 describe EITHER a redirect identifier or a ref identifier but not both.-->
3696       <xs:attribute name="redirect" type="xs:anyURI" use="optional"/>
3697       <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
3698     </xs:complexType>
3699   </xs:element>
3700 </xs:schema>
```

### 3703 **xrd.xsd**

```
3704 <?xml version="1.0" encoding="UTF-8"?>
3705 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3706 xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xrd="xri://$xrd*($v*2.0)"
3707 targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified">
3708   <!-- Utility patterns -->
3709   <xs:attributeGroup name="otherattribute">
3710     <xs:anyAttribute namespace="##other" processContents="lax"/>
3711   </xs:attributeGroup>
3712   <xs:group name="otherelement">
3713     <xs:choice>
3714       <xs:any namespace="##other" processContents="lax"/>
3715       <xs:any namespace="##local" processContents="lax"/>
3716     </xs:choice>
3717   </xs:group>
3718   <xs:attributeGroup name="priorityAttrGrp">
```

```

3719     <xs:attribute name="priority" type="xs:nonNegativeInteger" use="optional"/>
3720 </xs:attributeGroup>
3721 <xs:attributeGroup name="codeAttrGrp">
3722   <xs:attribute name="code" type="xs:int" use="required"/>
3723 </xs:attributeGroup>
3724 <xs:attributeGroup name="verifyAttrGrp">
3725   <xs:attribute name="cid" use="optional">
3726     <xs:simpleType>
3727       <xs:restriction base="xs:string">
3728         <xs:enumeration value="absent"/>
3729         <xs:enumeration value="off"/>
3730         <xs:enumeration value="verified"/>
3731         <xs:enumeration value="failed"/>
3732       </xs:restriction>
3733     </xs:simpleType>
3734   </xs:attribute>
3735   <xs:attribute name="ceid" use="optional">
3736     <xs:simpleType>
3737       <xs:restriction base="xs:string">
3738         <xs:enumeration value="absent"/>
3739         <xs:enumeration value="off"/>
3740         <xs:enumeration value="verified"/>
3741         <xs:enumeration value="failed"/>
3742       </xs:restriction>
3743     </xs:simpleType>
3744   </xs:attribute>
3745 </xs:attributeGroup>
3746 <xs:attributeGroup name="selectionAttrGrp">
3747   <xs:attribute name="match" use="optional" default="default">
3748     <xs:simpleType>
3749       <xs:restriction base="xs:string">
3750         <xs:enumeration value="default"/>
3751         <xs:enumeration value="any"/>
3752         <xs:enumeration value="non-null"/>
3753         <xs:enumeration value="null"/>
3754       </xs:restriction>
3755     </xs:simpleType>
3756   </xs:attribute>
3757   <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
3758 </xs:attributeGroup>
3759 <xs:attributeGroup name="appendAttrGrp">
3760   <xs:attribute name="append" use="optional" default="none">
3761     <xs:simpleType>
3762       <xs:restriction base="xs:string">
3763         <xs:enumeration value="none"/>
3764         <xs:enumeration value="local"/>
3765         <xs:enumeration value="authority"/>
3766         <xs:enumeration value="path"/>
3767         <xs:enumeration value="query"/>
3768         <xs:enumeration value="qxri"/>
3769       </xs:restriction>
3770     </xs:simpleType>
3771   </xs:attribute>
3772 </xs:attributeGroup>
3773 <xs:complexType name="URIPattern">
3774   <xs:simpleContent>
3775     <xs:extension base="xs:anyURI">
3776       <xs:attributeGroup ref="xrd:otherattribute"/>
3777     </xs:extension>
3778   </xs:simpleContent>
3779 </xs:complexType>
3780 <xs:complexType name="URIPriorityPattern">
3781   <xs:simpleContent>
3782     <xs:extension base="xrd:URIPattern">
3783       <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3784     </xs:extension>
3785   </xs:simpleContent>
3786 </xs:complexType>
3787 <xs:complexType name="URIPriorityAppendPattern">
3788   <xs:simpleContent>
3789     <xs:extension base="xrd:URIPriorityPattern">

```

```

3790         <xs:attributeGroup ref="xrd:appendAttrGrp" />
3791     </xs:extension>
3792 </xs:simpleContent>
3793 </xs:complexType>
3794 <xs:complexType name="StringPattern">
3795     <xs:simpleContent>
3796         <xs:extension base="xs:string">
3797             <xs:attributeGroup ref="xrd:otherattribute" />
3798         </xs:extension>
3799     </xs:simpleContent>
3800 </xs:complexType>
3801 <xs:complexType name="StringSelectionPattern">
3802     <xs:simpleContent>
3803         <xs:extension base="xrd:StringPattern">
3804             <xs:attributeGroup ref="xrd:selectionAttrGrp" />
3805         </xs:extension>
3806     </xs:simpleContent>
3807 </xs:complexType>
3808 <!-- Patterns for elements -->
3809 <xs:element name="XRD">
3810     <xs:complexType>
3811         <xs:sequence>
3812             <xs:element ref="xrd:Query" minOccurs="0" />
3813             <xs:element ref="xrd:Status" minOccurs="0" />
3814             <xs:element ref="xrd:ServerStatus" minOccurs="0" />
3815             <xs:element ref="xrd:Expires" minOccurs="0" />
3816             <xs:element ref="xrd:ProviderID" minOccurs="0" />
3817             <xs:choice>
3818                 <xs:element ref="xrd:Redirect" minOccurs="0" maxOccurs="unbounded" />
3819                 <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded" />
3820             </xs:choice>
3821             <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded" />
3822             <xs:element ref="xrd:EquipID" minOccurs="0" maxOccurs="unbounded" />
3823             <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded" />
3824             <xs:element ref="xrd:CanonicalEquipID" minOccurs="0"
3825 maxOccurs="unbounded" />
3826             <xs:element ref="xrd:Service" minOccurs="0" maxOccurs="unbounded" />
3827             <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded" />
3828         </xs:sequence>
3829         <xs:attribute name="idref" type="xs:IDREF" use="optional" />
3830         <xs:attribute name="version" type="xs:string" use="optional" fixed="2.0" />
3831         <xs:attributeGroup ref="xrd:otherattribute" />
3832     </xs:complexType>
3833 </xs:element>
3834 <xs:element name="Query" type="xrd:StringPattern" />
3835 <xs:element name="Status">
3836     <xs:complexType>
3837         <xs:simpleContent>
3838             <xs:extension base="xrd:StringPattern">
3839                 <xs:attributeGroup ref="xrd:codeAttrGrp" />
3840                 <xs:attributeGroup ref="xrd:verifyAttrGrp" />
3841                 <xs:attributeGroup ref="xrd:otherattribute" />
3842             </xs:extension>
3843         </xs:simpleContent>
3844     </xs:complexType>
3845 </xs:element>
3846 <xs:element name="ServerStatus">
3847     <xs:complexType>
3848         <xs:simpleContent>
3849             <xs:extension base="xrd:StringPattern">
3850                 <xs:attributeGroup ref="xrd:codeAttrGrp" />
3851                 <xs:attributeGroup ref="xrd:otherattribute" />
3852             </xs:extension>
3853         </xs:simpleContent>
3854     </xs:complexType>
3855 </xs:element>
3856 <xs:element name="Expires">
3857     <xs:complexType>
3858         <xs:simpleContent>
3859             <xs:extension base="xs:dateTime">
3860                 <xs:attributeGroup ref="xrd:otherattribute" />

```

```

3861         </xs:extension>
3862     </xs:simpleContent>
3863 </xs:complexType>
3864 </xs:element>
3865 <xs:element name="ProviderID" type="xrd:URIPattern"/>
3866 <xs:element name="LocalID">
3867     <xs:complexType>
3868         <xs:simpleContent>
3869             <xs:extension base="xrd:StringPattern">
3870                 <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3871             </xs:extension>
3872         </xs:simpleContent>
3873     </xs:complexType>
3874 </xs:element>
3875 <xs:element name="EquivID" type="xrd:URIPriorityPattern"/>
3876 <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>
3877 <xs:element name="CanonicalEquivID" type="xrd:URIPriorityPattern"/>
3878 <xs:element name="Redirect" type="xrd:URIPriorityAppendPattern"/>
3879 <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
3880 <xs:element name="Service">
3881     <xs:complexType>
3882         <xs:sequence>
3883             <xs:element ref="xrd:ProviderID" minOccurs="0"/>
3884             <xs:element ref="xrd:Type" minOccurs="0" maxOccurs="unbounded"/>
3885             <xs:element ref="xrd:Path" minOccurs="0" maxOccurs="unbounded"/>
3886             <xs:element ref="xrd:MediaType" minOccurs="0" maxOccurs="unbounded"/>
3887             <xs:choice>
3888                 <xs:element ref="xrd:URI" minOccurs="0" maxOccurs="unbounded"/>
3889                 <xs:element ref="xrd:Redirect" minOccurs="0" maxOccurs="unbounded"/>
3890                 <xs:element ref="xrd:Ref" minOccurs="0" maxOccurs="unbounded"/>
3891             </xs:choice>
3892             <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
3893             <xs:group ref="xrd:otherelement" minOccurs="0" maxOccurs="unbounded"/>
3894         </xs:sequence>
3895         <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
3896         <xs:attributeGroup ref="xrd:otherattribute"/>
3897     </xs:complexType>
3898 </xs:element>
3899 <xs:element name="Type">
3900     <xs:complexType>
3901         <xs:simpleContent>
3902             <xs:extension base="xrd:URIPattern">
3903                 <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
3904             </xs:extension>
3905         </xs:simpleContent>
3906     </xs:complexType>
3907 </xs:element>
3908 <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
3909 <xs:element name="Path" type="xrd:StringSelectionPattern"/>
3910 <xs:element name="URI" type="xrd:URIPriorityAppendPattern"/>
3911 </xs:schema>
3912

```

3913

---

## D. Media Type Definition for application/xrds+xml

3914 This section is prepared in anticipation of filing a media type registration meeting the  
3915 requirements of [RFC4288].

3916 **Type name:** application

3917 **Subtype name:** xrds+xml

3918 **Required parameters:** None

3919 **Optional parameters:** See Table 6 of this document.

3920 **Encoding considerations:** Identical to those of "application/xml" as described in [RFC3023],  
3921 Section 3.2.

3922 **Security considerations:** As defined in this specification. In addition, as this media type uses the  
3923 "+xml" convention, it shares the same security considerations as described in [RFC3023],  
3924 Section 10.

3925 **Interoperability considerations:** There are no known interoperability issues.

3926 **Published specification:** This specification.

3927 **Applications that use this media type:** Applications conforming to this specification use this  
3928 media type.

3929 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI  
3930 Technical Committee Co-Chair, drummond.reed@cordance.net

3931 **Intended usage:** COMMON

3932 **Restrictions on usage:** None

3933 **Author:** OASIS XRI TC

3934 **Change controller:** OASIS XRI TC

3935

---

## E. Media Type Definition for application/xrd+xml

3936 This section is prepared in anticipation of filing a media type registration meeting the  
3937 requirements of [RFC4288].

3938 **Type name:** application

3939 **Subtype name:** xrd+xml

3940 **Required parameters:** None

3941 **Optional parameters:** See Table 6 of this document.

3942 **Encoding considerations:** Identical to those of "application/xml" as described in [RFC3023],  
3943 Section 3.2.

3944 **Security considerations:** As defined in this specification. In addition, as this media type uses the  
3945 "+xml" convention, it shares the same security considerations as described in [RFC3023],  
3946 Section 10.

3947 **Interoperability considerations:** There are no known interoperability issues.

3948 **Published specification:** This specification.

3949 **Applications that use this media type:** Applications conforming to this specification use this  
3950 media type.

3951 **Person & email address to contact for further information:** Drummond Reed, OASIS XRI  
3952 Technical Committee Co-Chair, drummond.reed@cordance.net

3953 **Intended usage:** COMMON

3954 **Restrictions on usage:** None

3955 **Author:** OASIS XRI TC

3956 **Change controller:** OASIS XRI TC

3957

## F. Example Local Resolver Interface Definition (Informative)

3958

3959  
3960

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

3961  
3962  
3963  
3964  
3965

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.

3966

3967  
3968  
3969  
3970

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.
nodefault_t, nodefault_p, nodefault_m	Specifies whether a default match is allowed on the Type, Path, or MediaType elements respectively during service endpoint selection as defined in section 13.3.
uric	Specifies whether a resolver should automatically construct service endpoint URIs as defined in section 13.7.1.
cid	Specifies whether automatic canonical ID verification should performed as defined in section 14.3.

3971

3972  
3973  
3974  
3975

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

3976 Output Format is URI List, the `sep` subparameter MUST be considered to be TRUE, so there is  
3977 no `resolveAuthToURIList` operation.)  
3978

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

3979 Following is the API and descriptions of the five operations.

### 3980 1. Resolve Authority to XRDS

```
3981 Result resolveAuthToXRDS(  
3982     in string QXRI, in Flags flags);
```

- 3983 • Performs authority resolution only (sections 9 and 10) and outputs an XRDS document as  
3984 specified in section 8.2.1 when the `sep` subparameter is FALSE.
- 3985 • Only the authority component of the QXRI is processed by this function. If the QXRI contains  
3986 a path or query component, it is ignored.
- 3987 • Returns a binding-specific representation of the resolution result which may include, but is not  
3988 limited to, XRDS output, success/failure code, exceptions and error context.
- 3989 • The XRD element(s) in the output XRDS will be signed or not depending on the value of the  
3990 `saml` flag.

3991

### 3992 2. Resolve Authority to XRD

```
3993 Result resolveAuthToXRD(  
3994     in string QXRI, in Flags flags);
```

- 3995 • Performs authority resolution only (sections 9 and 10) and outputs an XRD element as  
3996 specified in section 8.2.2 when the `sep` subparameter is FALSE.
- 3997 • Only the authority component of the QXRI is processed by this function. If the QXRI contains  
3998 a path or query component, it is ignored.
- 3999 • Returns a binding-specific representation of the resolution result which may include, but is not  
4000 limited to, XRD output, success/failure code, exceptions and error context.
- 4001 • The output XRD will be signed or not depending on the value of the `saml` flag.

4002

### 4003 3. Resolve Service Endpoint to XRDS

```
4004 Result resolveSEPToXRDS(  
4005     in string QXRI, in string sepType,  
4006     in string sepMediaType, in Flags flags);
```

- 4007 • Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13)  
4008 and outputs the XRDS as specified in section 8.2.1 when the `sep` subparameter is TRUE.
- 4009 • Returns a binding-specific representation of the resolution result which may include, but is not  
4010 limited to, XRDS output, success/failure code, exceptions and error context.
- 4011 • The final XRD in the output XRDS will either contain at least one instance of the requested  
4012 service endpoint or an error. *IMPORTANT: Although the resolver will perform service  
4013 selection, the final XRD is NOT filtered when the Resolution Output Format is an XRDS  
4014 document. Filtering is only performed when the Resolution Output Format is an XRD  
4015 document (below).*
- 4016 • The XRD element(s) in the output XRDS will be signed or not depending on the value of  
4017 `saml` flag.

4018

### 4019 4. Resolve Service Endpoint to XRD

```
4020 Result resolveSEPToXRD(  
4021     in string QXRI, in string sepType,  
4022     in string sepMediaType, in Flags flags);
```

- 4023 • Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13)  
4024 and outputs an XRD as specified in section 8.2.2 when the `sep` subparameter is TRUE.
- 4025 • Returns a binding-specific representation of the resolution result which may include, but is not  
4026 limited to, XRD output, success/failure code, exceptions and error context.
- 4027 • The output XRD will contain at least one instance of the requested service endpoint or an  
4028 error. Also, all elements in the output XRD subject to the global `priority` attribute will be  
4029 returned in order of highest to lowest priority. See section 8.2.2 for details.
- 4030 • The XRD element will be signed or not depending on the value of `saml` flag, however that  
4031 signature may not be able to be independently verified because the XRD has been filtered to  
4032 contain only the selected service endpoints.

4033

4034 **5. Resolve Service Endpoint to URI List**

```
4035 Result resolveSepToURIList(  
4036     in string QXRI, in string sepType,  
4037     in string sepMediaType, in Flags flags);
```

- 4038
- 4039
- 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.
- 4040
- 4041
- 4042
- 4043
- 4044
- 4045

## G. Revision History

**Comment [DSR16]:** TODO-CD: We will revise this to reflect a condensed version history of XRI Resolution 2.0 as a whole.

Revision	Date	Editor	Changes Made
WD11 ED01	2007-05-23	Drummond Reed	All major changes from <a href="http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11">http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11</a> . A number of the more minor changes remain to be done.
WD11 ED02	2007-06-06	Drummond Reed	Added content of Appendix F and G prepared by Gabe Wachob. Moved "Discovery of XRDS Documents from HTTP URIs" to section 4, added overview, added extensive feedback. Fixed bug in section 9, Pseudocode (for Service Endpoint Selection) spotted by Markus Sabadello. Numerous minor errata identified by Gabe Wachob and Marcus Sabadello fixed. Added comments to section 11, CanonicalID Verification, indicating work to be done.
WD11 ED03	2007-07-24	Drummond Reed	Added section 2, Conformance (still needs to be completed). Revised section 5. Added section 7.1.4. Added section 9.8. Added new section 11, Synonyms. Renamed and rewrote section 12, Synonym Verification. 40+ other smaller changes as detailed on the XRI TC wiki at <a href="http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11">http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11</a> .
WD11 ED04	2007-09-06	Drummond Reed	Documented at <a href="http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11">http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11</a> .
WD11 ED05	2007-09-17	Drummond Reed	Documented at <a href="http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11">http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11</a> .
WD11 ED06	2007-10-16	Drummond Reed	Documented at <a href="http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11">http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11</a> .
WD11 ED07	2007-10-31	Drummond Reed	Revised frontmatter per TODO list. Reordered and revised appendices per instructions from Mary McRae (OASIS) Made revisions per email list discussions and TC telecon minutes – all normative changes have change marks.

			Added annotations to flowcharts and revised flowcharts Fig 5 and 8.
WD11 ED08	2007-11-07	Drummond Reed	<p>Revised text of sections 7 and 12.</p> <p>Replaced pseudocode in section 13.6 with Wil's compact version.</p> <p>Replaced RelaxNG schemas with Gabe's revised versions.</p> <p>Replaced Appendix F with Wil's and Steve's revisions and edited for consistency with ED08 references and terminology.</p>
WD11 RC1	2007-11-15	Drummond Reed	<p>Added section 5.4 and added/revised sentences in sections 5, 12, and 14 per input from John Bradley and Steve Churchill.</p> <p>Incorporated proofing feedback from Kermit Snelson.</p> <p>Fixed tabs in Appendix B and C.</p> <p>Checked and updated references throughout.</p> <p>Final proofreading pass.</p>

4048