Extensible Resource Identifier (XRI) Resolution Version 2.0

Committee Draft 03

28 February 2008

Specification URIs:

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

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

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

Technical Committee:
> OASIS eXtensible Resource Identifier (XRI) TC

Chairs:
> Gabe Wachob, AmSoft <gabe.wachob@amsoft.net>
> Drummond Reed, Cordance <drummond.reed@cordance.net>

Editors:
> Gabe Wachob, AmSoft <gabe.wachob@amsoft.net>
> Drummond Reed, Cordance <drummond.reed@cordance.net>
> Les Chasen, NeuStar <les.chasen@neustar.biz>
> William Tan, NeuStar <william.tan@neustar.biz>
> Steve Churchill, XDI.org <steven.churchill@xdi.org>

Related Work:
> This specification replaces or supercedes:

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

> This specification is related to:

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

Declared XML Namespace(s)
> xri://$res

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

Abstract:

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

Status:

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

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

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

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

Notices

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

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

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

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

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

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

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

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

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

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Extensible Resource Identifier (XRI) provides a uniform syntax for abstract structured identifiers as defined in **[XRISyntax]**. Because XRIs may be used across a wide variety of communities and applications (as Web addresses, database keys, filenames, object IDs, XML IDs, tags, etc.), no single resolution mechanism may prove appropriate for all XRIs. However, in the interest of promoting interoperability, this specification defines a simple generic resource description format called XRDS (Extensible Resource Descriptor Sequence), a standard protocol for requesting XRDS documents using HTTP(S) URIs, and standard protocol for resolving XRIs using XRDS documents and HTTP(S) URIs. Both generic and trusted versions of the XRI resolution protocol are defined (the latter using HTTPS **[RFC2818]** and/or signed SAML assertions **[SAML]**). In addition, an HTTP(S) proxy resolution service is specified both to provide network-based resolution services and for backwards compatibility with existing HTTP(S) infrastructure.

1.1 Overview of XRI Resolution Architecture

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

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

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

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

Table 1: Comparing DNS and XRI resolution architecture.

31 As Table 1 notes, XRI resolution architecture supports both recursing authority servers and *proxy*
32 *resolvers*. A proxy resolver is simply an HTTP(S) interface to a local XRI resolver (one
33 implemented using a platform-specific API). Proxy resolvers enable applications—even those that
34 only understand HTTP URIs—to easily access the functions of an XRI resolver remotely.

35 Figure 1 shows four scenarios of how these components might interact to resolve

36 `xri://(tel:+1-201-555-0123)*foo*bar` (unlike DNS, this works from left-to-right).

A) Local resolver

B) Local resolver using recursing authority server

C) Proxy resolver

D) Proxy resolver using recursing authority server

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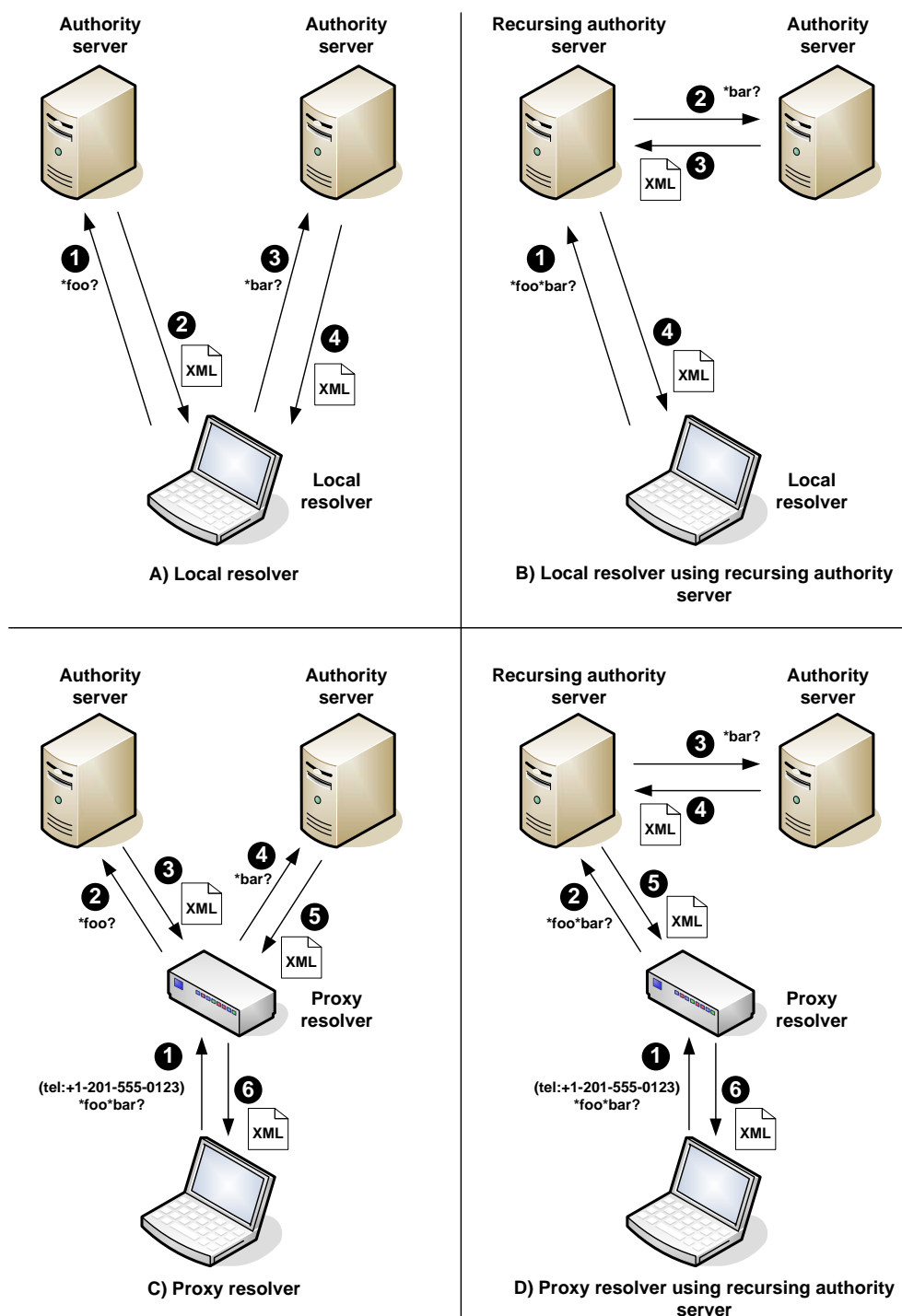
38   *Figure 1: Four typical scenarios for XRI authority resolution.*

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

40   • *Phase 1: Authority resolution.* This is the phase required to resolve the authority component
41   of an XRI into an XRDS document describing the target authority. Authority resolution works
42   iteratively from left-to-right across each subsegment in the authority component of the XRI. In
43   XRIs, subsegments are delimited using either a specified set of symbol characters or
44   parentheses. For example, in the XRI `xri://(tel:+1-201-555-0123)*foo*bar`, the
45   authority subsegments are `(tel:+1-201-555-0123)` (the community root authority, in this
46   case a URI expressed as an cross-reference delimited with parentheses), `*foo`, (the first
47   resolvable subsegment), and `*bar`, (the second resolvable subsegment). Note that a
48   resolver must be preconfigured (or have its own way of discovering) the community root
49   authority starting point, so the community root subsegment is not resolved except in one
50   special case (see section 9.1.6).

51   • *Phase 2: Optional service endpoint selection.* Once authority resolution is complete, there is
52   an optional second phase of XRI resolution to select a specific type of metadata from the final
53   XRDS document retrieved called a *service endpoint* (SEP). Service endpoints are descriptors
54   of concrete URIs at which network services are available for the target resource. Additional
55   XRI resolution parameters as well as the path component of an XRI may be used as service
56   endpoint selection criteria.

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

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

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

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

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

79   • *Redirects and Refs.* XRDS architecture also includes two mechanisms for distributed XRDS
80   document management. The *Redirect* element allows an identifier authority to manage
81   multiple XRDS documents describing a target resource from different network locations. The
82   *Ref* element allows one identifier authority to delegate all or part of an XRDS document to a
83   different identifier authority.

84

1.2 Structure of this Specification

This specification is structured into the following sections:

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

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

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

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

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

1.3 Terminology and Notation

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

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

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

Formatting conventions used in this document:

```
Examples look like this.
```

```
ABNF productions look like this.
```

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

1.4 Examples

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

1.5 Normative References

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

172 173	**[RFC4234]**	D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications: ABNF*, http://www.ietf.org/rfc/rfc4234.txt, IETF RFC 4234, October 2005.
174 175 176	**[RFC4288]**	N. Freed, J. Klensin, *Media Type Specifications and Registration Procedures*, http://www.ietf.org/rfc/rfc4288.txt, IETF RFC 4288, December 2005.
177 178 179 180	**[SAML]**	S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for the OASIS Security Assertion Markup Language* (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip, March 2005.
181 182 183 184 185	**[Unicode]**	The Unicode Consortium. The Unicode Standard, Version 4.1.0, defined by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), as amended by Unicode 4.0.1 (http://www.unicode.org/versions/Unicode4.0.1) and by Unicode 4.1.0 (http://www.unicode.org/versions/Unicode4.1.0), March, 2005.
186 187	**[UUID]**	Open Systems Interconnection – *Remote Procedure Call*, ISO/IEC 11578:1996, http://www.iso.org/, August 2001.
188 189 190	**[XML]**	T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, *Extensible Markup Language (XML) 1.0, Third Edition*, World Wide Web Consortium, http://www.w3.org/TR/REC-xml/, February 2004.
191 192 193	**[XMLDSig]**	D. Eastlake, J. Reagle, D. Solo et al., *XML-Signature Syntax and Processing*, World Wide Web Consortium, http://www.w3.org/TR/xmldsig-core/, February, 2002.
194 195 196	**[XMLID]**	J. Marsh, D. Veillard, N. Walsh, *xml:id Version 1.0*, World Wide Web Consortium, http://www.w3.org/TR/2005/REC-xml-id-20050909, September 2005.
197 198 199	**[XMLSchema]**	H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, *XML Schema Part 1: Structures Second Edition*, World Wide Web Consortium, http://www.w3.org/TR/xmlschema-1/, October 2004.
200 201 202	**[XMLSchema2]**	P. Biron, A. Malhotra, *XML Schema Part 2: Datatypes Second Edition*, World Wide Web Consortium, http://www.w3.org/TR/xmlschema-2/, October 2004.
203 204 205	**[XRIMetadata]**	D. Reed, *Extensible Resource Identifier (XRI) Metadata V2.0*, http://docs.oasis-open.org/xri/xri/V2.0/xri-metadata-V2.0-cd-01.pdf, March 2005.
206 207 208	**[XRISyntax]**	D. Reed, D. McAlpin, *Extensible Resource Identifier (XRI) Syntax V2.0*, http://docs.oasis-open.org/xri/xri-syntax/2.0/specs/cs01/xri-syntax-V2.0-cs.pdf, November 2005.

209   1.6 Non-Normative References

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

2 Conformance

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

2.1 Conformance Targets

The conformance targets of this specification are:

1. *XRDS clients*, which provide a limited subset of the functionality of XRI resolvers.
2. *XRDS servers*, which provide a limited subset of the functionality of XRI authority servers.
3. *XRI local resolvers*, which may implement any combination of the generic, HTTPS, or SAML resolution protocols.
4. *XRI proxy resolvers*, which may implement any combination of the generic, HTTPS, or SAML resolution protocols.
5. *XRI authority servers*, which may implement any combination of the generic, HTTPS, or SAML resolution protocols.

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

2.2 Conformance Claims

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

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

2.3 XRDS Clients

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

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

2.4 XRDS Servers

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

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

2.5 XRI Local Resolvers

2.5.1 Generic

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

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

2.5.2 HTTPS

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

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

2.5.3 SAML

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

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

2.6 XRI Proxy Resolvers

2.6.1 Generic

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

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

2.6.2 HTTPS

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

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

2.6.3 SAML

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

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

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

293 2.7 XRI Authority Servers

294 2.7.1 Generic

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

    297   1. It produces XRDS documents as specified in section 4.

    298   2. It assigns XRDS synonyms as specified in section 5.

    299   3. It processes resolution inputs and outputs as specified in section 8.

    300   4. It implements the server requirements of the generic resolution protocol specified in
301   section 9.

    302   5. It implements the Status Code and Error Processing rules specified in section 15.

    303   6. It follows the HTTP(S) usage recommendations specified in section 16.

304 2.7.2 HTTPS

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

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

309 2.7.3 SAML

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

    312   1. It implements the server requirements of the SAML trusted resolution protocol specified
313   in section 10.2.

    314   2. It SHOULD also meet the requirements of an HTTPS authority server. This is
315   STRONGLY RECOMMENDED for confidentiality of SAML interactions.

316 2.8 Extensions

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

321 2.9 Language

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

324 3 Namespaces

325 3.1 XRI Namespaces for XRI Resolution

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

330 3.1.1 XRIs Reserved for XRI Resolution

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

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

333 *Table 2: XRIs reserved for XRI resolution.*

334 3.1.2 XRIs Assigned to XRI Resolution Service Types

335 The XRIs in Table 3 are assigned to the XRI resolution service types defined in this specification.

XRI	Usage	See Section
xri://$res*auth	Authority resolution service	9
xri://$res*auth*($v*2.0)	Version 2.0 of above	9
xri://$res*proxy	HTTP(S) proxy resolution service	11
xri://$res*proxy*($v*2.0)	Version 2.0 of above	11

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

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

340

3.2 XML Namespaces for XRI Resolution

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

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

Table 4: XML namespace prefixes used in this specification.

3.3 Media Types for XRI Resolution

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

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

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

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

358

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

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

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

362 4 XRDS Documents

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

368 4.1 XRDS and XRD Namespaces and Schema Locations

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

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

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

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

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

387 • **xrds.rnc**: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrds.mc

388 • **xrd.rnc**: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrd.mc

389 •

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

391 • **xrds.rnc**: http://docs.oasis-open.org/xri/2.0/specs/xrds.rnc

392 • **xrd.rnc**: http://docs.oasis-open.org/xri/2.0/specs/xrd.rnc

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

395 4.2 XRD Elements and Attributes

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

400

```
401

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

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

461

4.2.1 Management Elements

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

xrd:XRD

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

xrd:XRD/xrd:Type

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

xrd:XRD/xrd:Query

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

xrd:XRD/xrd:Status

> 0 or 1 per `xrd:XRD` element. RECOMMENDED for all XRDs. REQUIRED if the resolver must report certain error conditions. The contents of the element are a human-readable message string describing the status of the response as determined by the resolver. For XRI resolution, values of the Status element are defined in section 15. Attributes:
>
> - `code` (type `xs:int`). REQUIRED. Provides a numeric status code. See section 15.
>
> - `cid` (type `xs:enumeration`). OPTIONAL except when REQUIRED to report the results of CanonicalID verification as defined in section 14.3.4.
>
> - `ceid` (type `xs:enumeration`). OPTIONAL except when REQUIRED to report the results of CanonicalID verification as defined in section 14.3.4.

xrd:XRD:xrd:ServerStatus

> 0 or 1 per `xrd:XRD` element. Used by an XRI authority server to report the status of a resolution request to an XRI resolver. See section 15.1. Attributes:
>
> - `code` (type `xs:int`). REQUIRED. Provides a numeric status code. See section 15.

xrd:XRD/xrd:Expires

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

xrd:XRD/xrd:Redirect

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

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

> - `append` (type `xs:enumeration`). OPTIONAL. Governs construction of the final redirect URI as defined in section 13.7.

xrd:XRD/xrd:Ref

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

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

4.2.2 Trust Elements

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

xrd:XRD/xrd:ProviderID

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

xrd:XRD/saml:Assertion

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

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

4.2.3 Synonym Elements

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

xrd:XRD/xrd:LocalID

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

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

xrd:XRD/xrd:EquivID

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

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

xrd:XRD/xrd:CanonicalID

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

xrd:XRD/xrd:CanonicalEquivID

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

4.2.4 Service Endpoint Descriptor Elements

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

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

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

xrd:XRD/xrd:Service

596 0 or more per `xrd:XRD` element. The container element for service endpoint metadata.
597 Referred to by the abbreviation *SEP*. Attributes:

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

xrd:XRD/xrd:Service/xrd:LocalID

600 0 or more per `xrd:XRD/xrd:Service` element. Identical to the
601 `xrd:XRD/xrd:LocalID` element defined above except this synonym is asserted by the
602 provider of the service and not the parent authority for the XRD. MAY be used to provide
603 one or more identifiers by which the target resource SHOULD be identified in the context
604 of the service endpoint. See section 5.2.1 for detailed requirements. Attributes:

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

xrd:XRD/xrd:Service/xrd:URI

607 0 more per `xrd:XRD/xrd:Service` element. Type `xs:anyURI`. Choice between this or
608 the `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
609 elements. If present, it indicates a transport-level URI for accessing the capability
610 described by the parent Service element. For the service types defined for XRI resolution
611 in section 3.1.2, this URI MUST be an HTTP or HTTPS URI. Other services may use
612 other transport protocols. Attributes:

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

614 • `append` (type `xs:enumeration`). OPTIONAL. Governs construction of the final
615 service endpoint URI as defined in section 13.7.

xrd:XRD/xrd:Service/xrd:Redirect

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

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

xrd:XRD/ xrd:Service/xrd:Ref

623 0 more per `xrd:XRD/xrd:Service` element. Choice between this or the
624 `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Redirect`
625 elements. Identical to the `xrd:XRD/xrd:Ref` element defined above except processed
626 only in the context of service endpoint selection. See section 12. Attributes:

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

628

4.2.5 Service Endpoint Trust Elements

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

xrd:XRD/xrd:Service/xrd:ProviderID

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

xrd:XRD/xrd:Service/ds:KeyInfo

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

4.2.6 Service Endpoint Selection Elements

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

xrd:XRD/xrd:Service/xrd:Type

> 0 or more per `xrd:XRD/xrd:Service` element. A unique identifier of type `xs:anyURI` that identifies the type of capability available at this service endpoint. See section 3.1.2 for the resolution service types defined in this specification. If a service endpoint does not include at least one `xrd:Type` element, the service type is effectively described by the type of URI specified in the `xrd:XRD/xrd:Service/xrd:URI` element, i.e., an HTTP URI specifies an HTTP service. See section 13.3.6 for Type element matching rules. Attributes:
>
> - `match` (type `xs:enumeration`). OPTIONAL. See section 13.3.2.
> - `select` (type `xs:boolean`). OPTIONAL. See section 13.4.2.

xrd:XRD/xrd:Service/xrd:Path

> 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string meeting the `xri-path` production defined in section 2.2.3 of **[XRISyntax]**. See section 13.3.7 for Path element matching rules. Attributes:
>
> - `match` (type `xs:enumeration`). OPTIONAL. See section 13.3.2.
> - `select` (type `xs:boolean`). OPTIONAL. See section 13.4.2.

xrd:XRD/xrd:Service/xrd:MediaType

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

| 673 | of a media type defined in **[RFC2046]**. See section 3.3 for the media types used in XRI |
| 674 | resolution. See section 13.3.8 for MediaType element matching rules. Attributes: |

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

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

4.3 XRD Attribute Processing Rules
681

4.3.1 ID Attribute
682

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

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

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

4.3.2 Version Attribute
701

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

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

4.3.3 Priority Attribute
708

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

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

    1. The consuming application SHOULD select the element instance with the lowest numeric
       value of the `priority` attribute. For example, an element with `priority` attribute value
       of "10" should be selected before an element with a `priority` attribute value of "11",
       and an element with `priority` attribute value of "11" should be selected before an
       element with a `priority` attribute value of "25". Zero is the highest `priority` attribute
       value. Null is the lowest `priority` attribute value—it is the equivalent of a value of
       infinity. It is RECOMMENDED to use a large finite value (100 or more) rather than a null
       value.

    2. If an element has no `priority` attribute, its `priority` attribute value is considered to
       be null, i.e., the lowest possible priority value. Rather than omitting a `priority` attribute,
       it is RECOMMENDED that XRI authorities follow the standard practice in DNS and set
       the default `priority` attribute value to "10".

    3. If two or more instances of the same element type have identical `priority` attribute
       values (including the null value), the consuming application SHOULD select one of the
       instances at random. This consuming application SHOULD NOT simply choose the first
       instance that appears in XML document order.

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

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

4.4 XRI and IRI Encoding Requirements

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

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

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

761 5 XRD Synonym Elements

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

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

765 *Table 7: The four XRD synonym elements.*

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

767 5.1 Query Identifiers

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

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

772   2. A valid absolute XRI, either in a standard XRI form as defined in **[XRISyntax]**, or
773   encoded in an HTTP(S) URI (called an *HXRI*) as specified in section 11.2.

774 5.1.1 HTTP(S) URI Query Identifiers

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

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

781 5.1.2 XRI Query Identifiers

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

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

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

5.2 Synonym Elements

5.2.1 LocalID

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

800 At the XRD level, the value of the `xrd:XRD/xrd:LocalID` element asserts a synonym that is
801 interchangeable with the contents of the `xrd:Query` element in the XRD. This means that
802 resolution of a LocalID in the context of the same parent authority using the same resolution
803 query parameters as the current query MUST result in an equivalent XRD as defined in section
804 5.4. It also means an XRI resolver MAY use a LocalID as an alternate key for the XRD in its
805 cache (see section 16.4.2).

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

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

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

5.2.2 EquivID

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

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

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

828   1.   When it is REQUIRED as a backpointer to verify another EquivID element in a different
829   XRD as specified in section 14.2.

830   2.   When it is REQUIRED as a backpointer to verify a CanonicalEquivID element as
831   specified in section 14.3.3.

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

5.2.3 CanonicalID

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

845   1. It MUST be an identifier for which the parent authority is the final authority. This means
846   that resolution of a CanonicalID using the same resolution query parameters as the
847   current query MUST result in an equivalent XRD as defined in section 5.4.

848   2. If the CanonicalID is any XRI except a community root authority XRI (section 9.1.3), it
849   MUST consist of the parent authority's CanonicalID plus one additional subsegment. (In
850   XRI resolution the parent authority's CanonicalID is always in the immediately preceding
851   XRD in the same XRDS document, not in a nested XRDS document produced as a result
852   of Redirect and Ref processing as defined in section 12.5.) For example, if the
853   CanonicalID asserted for a target resource is `@!1!2!3`, then the CanonicalID for the
854   parent authority must be `@!1!2`. See section 14.3.2 for details.

855   3. Once assigned, a parent authority SHOULD NEVER: a) change or reassign a
856   CanonicalID value, or b) stop asserting a CanonicalID element in an XRD in which it has
857   been asserted. For this reason, it is STRONGLY RECOMMENDED to use a persistent
858   identifier such as a persistent XRI **[XRISyntax]** or a URN **[RFC2141]**.

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

861   •   Makes it unambiguous to consuming applications which absolute synonym they should use to
862   identify the target resource in the context of the parent authority.

863   •   Enables child authorities to issue their own verifiable CanonicalIDs.

864   •   Enables verification of a CanonicalEquivID if asserted (below).

5.2.4 CanonicalEquivID

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

875   1. In order for the value of the `xrd:CanonicalEquivID` element to be verified: a) the
876   XRD in which it appears MUST include a CanonicalID that can be verified as specified in
877   section 14.2, and b) the XRD to which it resolves MUST meet the rules specified in
878   section 14.3.3. In particular, those rules require that the CanonicalID of that XRD match
879   the asserted CanonicalEquivID.

880   2. For the same reasons as with a CanonicalID, it is STRONGLY RECOMMENDED to use
881   a persistent identifier such as a persistent XRI **[XRISyntax]** or a URN **[RFC2141]**.

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

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

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

5.3 Redirect and Ref Elements

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

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

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

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

5.4 XRD Equivalence

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

919 1. Both XRDs contain a CanonicalID element.

920 2. The values of these CanonicalID elements are equivalent according to the equivalence
921   rules of the applicable identifier scheme. Note that these identifiers MUST be in URI-
922   normal form as specified in section 4.4. In addition, if the CanonicalID values are
923   HTTP(S) URIs, fragments MUST be considered significant in comparison.

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

5.5 Synonym Verification

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

5.6 Synonym Selection

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

- Only select a verified synonym (see above).

- Select a persistent synonym, particularly if a long term or immutable reference is required. If a persistent synonym is present, other reassignable synonyms (including the current fully-qualified query identifier) SHOULD be treated only as temporary identifiers.

- Select a CanonicalID if present, verified, and persistent. This identifier SHOULD be used whenever referencing the target resource in the context of the parent authority issuing the CanonicalID.

- If possible, *also* select a CanonicalEquivID if present, verified, and persistent. This identifier SHOULD be used as a reference to the target resource in any context other than that of the parent authority.

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

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

949
950

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

6.1 Overview

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

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

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

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

6.2 HEAD Protocol

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

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

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

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

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

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

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

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

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

6.3 GET Protocol

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

986 The XRDS server response MUST be one of four options:

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

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

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

997 If the response is only an HTML document as defined in the third option above, the value of the
998 `<meta>` element with an `http-equiv` attribute equal to `X-XRDS-Location` MUST be an
999 HTTP(S) URI which gives the location of an XRDS document describing the target resource. If
1000 this HTTP(S) URI is identical to the starting HTTP(S) URI, this is a loop and the protocol fails.
1001 Otherwise, the XRDS client MUST request the XRDS document from this URI using an HTTP(S)
1002 GET. This request SHOULD include an Accept header specifying the content type
1003 `application/xrds+xml`.

1004 If the response includes both an HTTP(S) response header and the HTML document defined in
1005 the third option above, the value of the `X-XRDS-Location` element in the HTTP(S) response-
1006 header MUST take precedence.

1007 If the response includes an XRDS document as specified in the fourth option above, the protocol
1008 has completed successfully.

1009 In all cases the HTTP(S) status messages and error codes defined in **[RFC2616]** apply.

1010 Note: If the XRDS server supports content negotiation, the response SHOULD include a `Vary:`
1011 header to allow caches to properly interpret future requests. This header SHOULD be present
1012 even in the case where the HTML page is returned (instead of an XRDS document).

1013 # 7 XRI Resolution Flow

1014 Logically, XRI resolution is a function invoked by an application to dereference an XRI into a
1015 descriptor of the target resource (or in some cases to a representation of the resource itself).
1016 Figure 2 is a top-level flowchart of this function highlighting the two major phases: *authority*
1017 *resolution* followed by *optional service endpoint selection*.
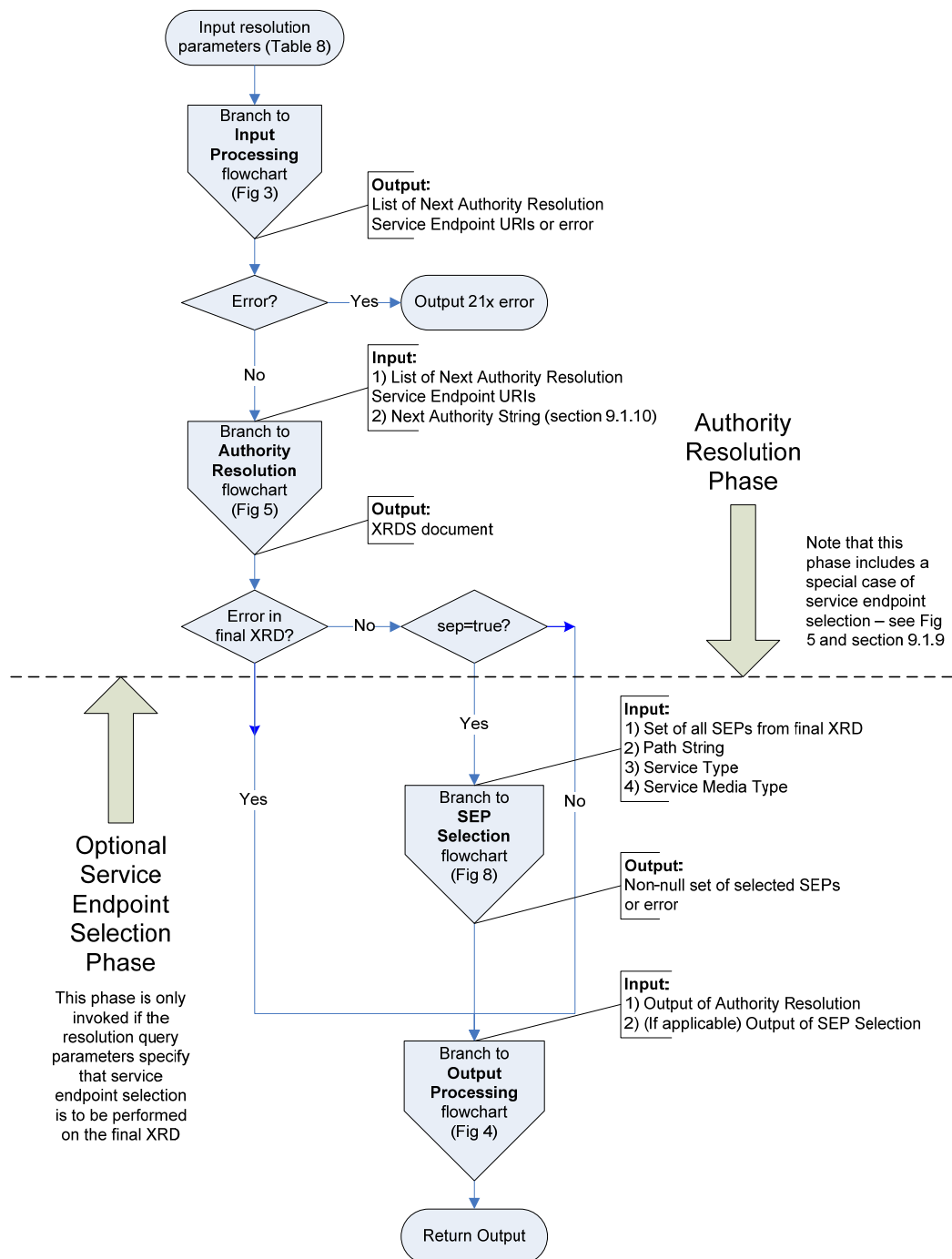
1018
1019 *Figure 2: Top-level flowchart of XRI resolution phases.*

1020 Branches of this top-level flowchart are used throughout the specification to provide a logical
1021 overview of key components of XRI resolution. The branch flowcharts include:

1022 • Figure 3: Input processing (section 8.1).

1023 • Figure 4: Output processing (section 8.2).

1024 • **Figure 5: Authority resolution (section 9).**

1025 • Figure 6: XRDS requests (section 9.1.3).

1026 • **Figure 7: Redirect and Ref processing (section 12).**

1027 • **Figure 8: Service endpoint selection (section 13).**

1028 • Figure 9: Service endpoint selection logic (section 13.2).

1029 IMPORTANT: In all cases the flowcharts are informative and the specification text is normative.
1030 However, the flowcharts are recommended as an aid in reading the specification. In particular,
1031 those highlighted in **bold** above illustrate the recursive calls for authority resolution and service
1032 endpoint selection used during Redirect and Ref processing (section 12). Implementers should
1033 pay special attention to these calls and the guidance in section 12.6, *Recursion and Backtracking*.

₁₀₃₄ # 8 Inputs and Outputs

₁₀₃₅ This section defines the logical inputs and outputs of XRI resolution together with their processing
₁₀₃₆ rules. It does not specify a binding to a particular local resolver interface. A binding to an HTTP
₁₀₃₇ interface for XRI proxy resolvers is specified in section 11. For purposes of illustration, a binding
₁₀₃₈ to a non-normative, language-neutral API is suggested in Appendix F.

₁₀₃₉ ## 8.1 Inputs

₁₀₄₀ Table 8 summarizes the logical input parameters to XRI resolution and whether they are
₁₀₄₁ applicable in the authority resolution phase or the service endpoint selection phase. In this
₁₀₄₂ specification, references to these parameters use the logical names in the first column. Local
₁₀₄₃ 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

₁₀₄₄ *Table 8: Input parameters for XRI resolution.*

₁₀₄₅ The following general rules apply to all input parameters as well as to all XRD elements
₁₀₄₆ throughout this specification:

₁₀₄₇ 1. The presence of an input parameter, subparameter, or XRD element with an empty value
₁₀₄₈ MUST be treated as equivalent to the absence of that input parameter, subparameter, or
₁₀₄₉ XRD element. (Note that this rule does not apply to XRD attributes.)

₁₀₅₀ 2. From a programmatic standpoint, both conditions above MUST be considered as
₁₀₅₁ equivalent to setting the value of that parameter, subparameter, or element to null.

₁₀₅₂ 3. In an XRD element, an attribute with an empty value is an error and MUST NOT be
₁₀₅₃ interpreted as the default value or any other value of that attribute.

₁₀₅₄ 4. As required by **[XMLSchema2]**, for all Boolean subparameters: a) the string values `true`
₁₀₅₅ and `false` MUST be considered case-insensitive (lowercase is RECOMMENDED), b)
₁₀₅₆ the values `true` and `1` MUST be considered equivalent, b) the values `false` and `0`
₁₀₅₇ MUST be considered equivalent.

₁₀₅₈

1059   Figure 3 is a flowchart (non-normative) illustrating the processing of input parameters.

```
                    ┌──────────────┐
                    │ Begin Input  │
                    │  Processing  │──┐ ┌─────────────────────────────┐
                    └──────────────┘  └─│ Input:                      │
                            │           │ Resolution parameters (Table 8) │
                            ▼           └─────────────────────────────┘
                      ◇ QXRI valid? ◇───No──→ ┌──────────────────────────┐
                            │                 │ Output error 210 or 211  │
                           Yes                └──────────────────────────┘
                            ▼
                      ◇   QXRI   ◇
                      includes XRI
                      resolution para-
                         meters?
                            │
                           Yes
                            ▼
                      ◇  Valid   ◇──────No──→ ┌──────────────────────────┐
         No           parameters?            │ Output errors 212, 213, or 214 │
                            │                 └──────────────────────────┘
                           Yes
                            ▼
                      ◇  Known   ◇
                      community root ──No──→ ┌──────────────────────────┐
                       authority?           │    Output error 216      │
                            │                └──────────────────────────┘
                           Yes
                            ▼
              ┌───────────────────────────┐   ┌──────────────────────────────┐
              │ Select authority resolution│──→│       Output list of         │
              │   service endpoints for    │   │ Next Authority Resolution Service │
              │  community root authority  │   │  Endpoint URIs (section 9.1.10)  │
              └───────────────────────────┘   └──────────────────────────────┘
```

1060

1061   *Figure 3: Input processing flowchart.*

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

1064

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

1065

1066 The QXRI (query XRI) is the only REQUIRED input parameter. Per **[XRISyntax]**, a QXRI consists
1067 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.

1068 *Table 9: Subparameters of the QXRI input parameter.*

1069 The fourth possible component of a QXRI—a fragment—is by definition resolved locally relative
1070 to the target resource identified by the combination of the Authority, Path, and Query
1071 components, and as such does not play a role in XRI resolution.

1072 Following are the constraints on the value of the QXRI parameter.

1073   1. It MUST be a valid absolute XRI according to the ABNF defined in **[XRISyntax]**. To
1074   resolve a relative XRI reference, it must be converted into an absolute XRI using the
1075   procedure defined in section 2.4 of **[XRISyntax]**.

1076   2. For authority or proxy resolution as defined in this specification, the QXRI MUST be in
1077   URI-normal form as defined in section 2.3.1 of **[XRISyntax]**. A local resolver API MAY
1078   support the input of other XRI forms but SHOULD document the normal form(s) it
1079   supports and its normalization policies.

1080   3. When a QXRI is included as part of an HXRI (section 11.2) for XRI proxy resolution, the
1081   QXRI MUST be normalized as specified in section 11.2, and all HXRI query parameters
1082   MUST follow the encoding rules specified in sections 11.3 and 11.4.

8.1.2 Resolution Output Format

1083

1084 The Resolution Output Format is an OPTIONAL parameter that, together with its subparameters,
1085 is used to specify:

1086   •   The media type for the resolution response.

1087   •   Whether generic or trusted resolution must be used by the resolver.

1088   •   Whether Refs should be followed during resolution.

1089   •   Whether CanonicalID verification should not be performed during resolution.

1090   •   Whether service endpoint selection should be performed on the final XRD.

1091 • Whether default matches should be ignored during service endpoint selection.

1092 • Whether URIs should automatically be constructed in the final XRD.

1093 Following are the normative requirements for the use of this parameter.

1094 1. The Resolution Output Format MUST be one of the values specified in Table 5 and MAY
1095 include any of the subparameters specified in Table 6.

1096 2. If the value of the `https` subparameter is TRUE, the resolver MUST use the HTTPS
1097 trusted authority resolution protocol specified in section 10.1 (or return an error indicating
1098 this is not supported).

1099 3. If the value of the `saml` subparameter is TRUE, the resolver MUST use the SAML trusted
1100 authority resolution protocol specified in section 10.2 (or return an error indicating this is
1101 not supported).

1102 4. If the value of both the `https` and `saml` subparameters are TRUE, the resolver MUST
1103 use the HTTPS+SAML trusted authority resolution protocol specified in section 10.3 (or
1104 return an error indicating this is not supported).

1105 5. If the value of the `cid` subparameter is TRUE or null, or if the parameter is absent, the
1106 resolver MUST perform CanonicalID verification as specified in section 14.3. If the value
1107 of the `cid` subparameter is FALSE, the resolver MUST NOT perform CanonicalID
1108 verification.

1109 6. If the value of the `refs` subparameter is TRUE or null, or if the parameter is absent, the
1110 resolver MUST perform Ref processing as specified in section 12. If the value of the
1111 `refs` subparameter is FALSE, the resolver MUST NOT perform Ref processing and
1112 must return an error if a Ref is encountered as specified in section 12.

1113 7. If the value of the `sep` subparameter is TRUE, the resolver MUST perform service
1114 endpoint selection on the final XRD (even if the values of all service endpoint selection
1115 parameters are null). If the value of the `sep` subparameter is FALSE or null, or if the
1116 parameter is absent, the resolver MUST NOT perform service endpoint selection on the
1117 final XRD unless it is required to produce a URI List or HTTP(S) redirect. See section 8.2.

1118 8. If the value of the `nodefault_t`, `nodefault_p`, or `nodefault_m` subparameter is
1119 TRUE, the resolver MUST ignore default matches on the corresponding service endpoint
1120 selection element categories as specified in section 13.3.2.

1121 9. If the value of the `uric` subparameter is TRUE, the resolver MUST perform service
1122 endpoint URI construction as specified in section 13.7.1. If the value of the `uric`
1123 subparameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT
1124 perform service endpoint URI construction.

1125 Future versions of this specification, or other specifications for XRI resolution, MAY use other
1126 values for Resolution Output Format or its subparameters.

1127 8.1.3 Service Type

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

8.1.4 Service Media Type

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

8.2 Outputs

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

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

Table 10: Outputs of XRI resolution.

1149 Figure 4 is a flowchart illustrating the process of producing these output formats once the auth-
1150 ority resolution and optional service endpoint selection phases are complete. Note that in the first
1151 two output options, errors are reported directly in the XRDS, so no special error format is needed.

1152
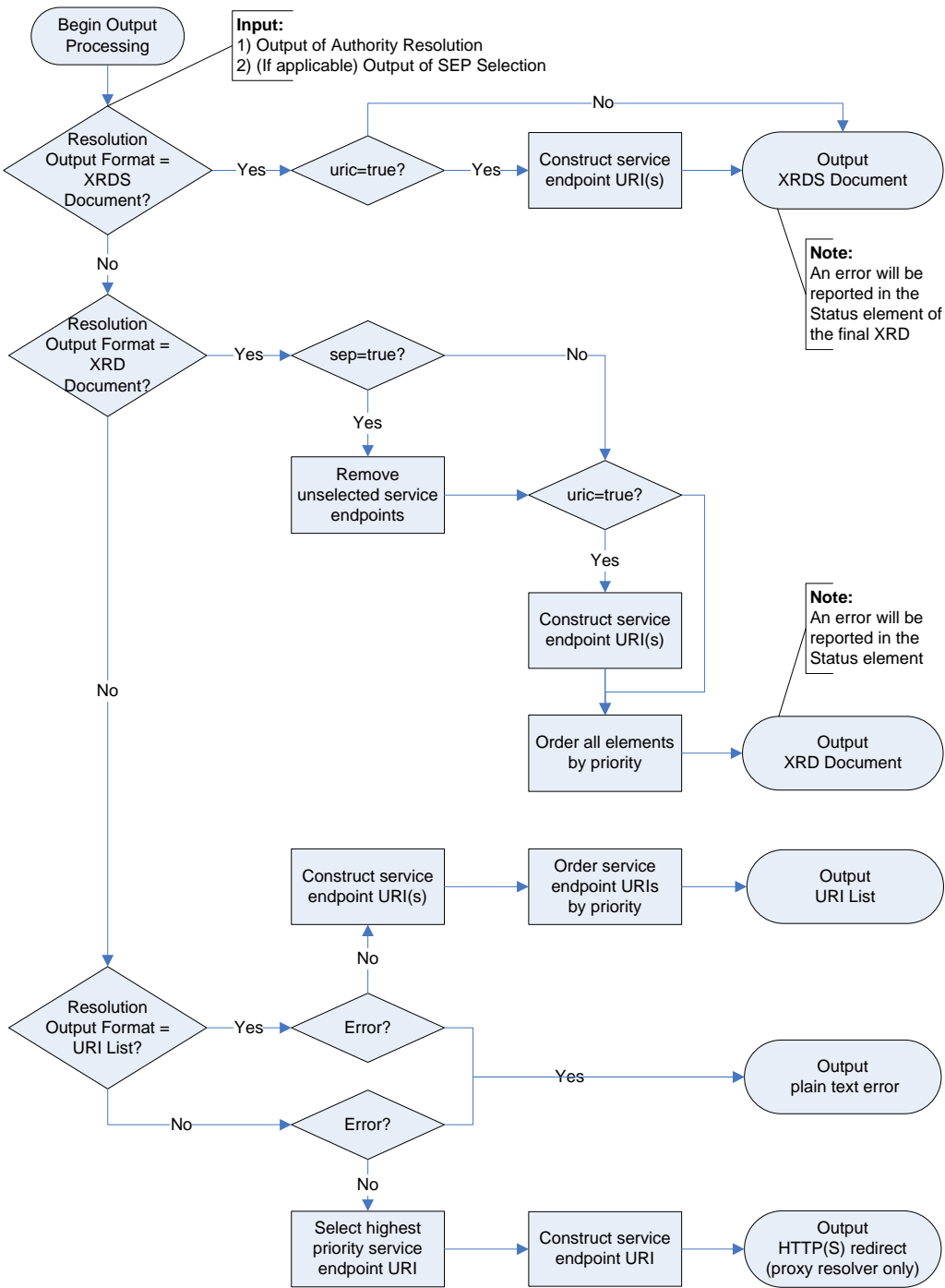
1153 *Figure 4: Output processing flowchart.*

1154 The following sections provide additional construction and validation requirements.

1155

8.2.1 XRDS Document

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

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

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

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

8.2.2 XRD Element

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

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

1201	present. In addition, all elements within the XRD element that are subject to the global
1202	`priority` attribute (even if the attribute is absent or null) MUST be returned in order of
1203	highest to lowest priority as defined in section 4.3.3.

1204 IMPORTANT: Any other filtering of the XRD element MUST NOT be performed. Note that this
1205 means that if the XRD element includes a SAML signature element as defined in section 10.2,
1206 this element is still returned inside the XRD element even though it may not be able to be verified
1207 by a consuming application.

1208   7. If the value of the `cid` subparameter is TRUE, synonym verification MUST be reported
1209   using the `xrd:Status` element of each XRD in the XRDS document as defined in
1210   section 14.

1211   8. If the output is an error, this error MUST be returned using the `xrd:Status` element as
1212   defined in section 15.

1213 8.2.3 URI List

1214 If the value of the Resolution Output Format parameter is `text/uri-list`, the following rules
1215 apply.

1216   1. For this output, service endpoint selection is REQUIRED, even if the values of all service
1217   endpoint selection input parameters are null.

1218   2. If authority resolution and service endpoint selection are both successful, the output
1219   MUST be a valid URI List as defined by section 5 of **[RFC2483]**.

1220   3. If, after applying the service endpoint selection rules, more than one service endpoint is
1221   selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as
1222   defined in section 4.3.3.

1223   4. If the final selected `xrd:XRD/xrd:Service` element contains a
1224   `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
1225   element, Redirect and Ref processing MUST be performed as described in section 12.
1226   This rule applies iteratively to each new XRDS document resolved.

1227   5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s)
1228   MUST be constructed as defined in section 13.7.1.

1229   6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI`
1230   elements within the selected `xrd:Service` element as defined in section 4.3.3. When
1231   two or more of the source `xrd:URI` elements have equal priority, their constructed URIs
1232   SHOULD be returned in random order.

1233 IMPORTANT: Any other filtering of the URI list MUST NOT be performed.

1234   7. If the output is an error, it MUST be returned with the content type `text/plain` as
1235   defined in section 15.

1236 8.2.4 HTTP(S) Redirect

1237 In XRI proxy resolution, the Resolution Output Format parameter may be null. In this case the
1238 output of a proxy resolver is an HTTP(S) redirect as defined in section 11.7.

¹²³⁹ 9 Generic Authority Resolution Service

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

¹²⁴⁴ XRI authorities and IRI authorities have different syntactic structures, partially due to the higher
¹²⁴⁵ level of abstraction represented by XRI authorities. For this reason, XRI authorities are resolved
¹²⁴⁶ to XRDS documents one subsegment at a time as specified in section 9.1. IRI authorities, since
¹²⁴⁷ they are based on DNS names or IP addresses, are resolved into an XRDS document through a
¹²⁴⁸ special HTTP(S) request using the entire IRI authority component as specified in section 9.1.11.

¹²⁴⁹ 9.1 XRI Authority Resolution

¹²⁵⁰ 9.1.1 Service Type and Service Media Type

¹²⁵¹ The protocol defined in this section is identified by the values in Table 11.

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

¹²⁵² *Table 11: Service Type and Service Media Type values for generic authority resolution.*

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

¹²⁵⁵ BACKWARDS COMPATIBILITY NOTE: Earlier drafts of this specification used a subparameter
¹²⁵⁶ called `trust`. This has been deprecated in favor of new subparameters for each trusted
¹²⁵⁷ resolution option, i.e., `https=true` and `saml=true`. However, implementations SHOULD
¹²⁵⁸ consider the following values equivalent both for the purpose of service endpoint selection within
¹²⁵⁹ XRDS documents and as HTTP(S) Accept header values in XRI authority resolution requests:
¹²⁶⁰   `application/xrds+xml`
¹²⁶¹   `application/xrds+xml;trust=none`
¹²⁶²   `application/xrds+xml;https=false`
¹²⁶³   `application/xrds+xml;saml=false`
¹²⁶⁴   `application/xrds+xml;https=false;saml=false`
¹²⁶⁵   `application/xrds+xml;saml=false;https=false`

¹²⁶⁶

1267   ## 9.1.2 Protocol

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

1269
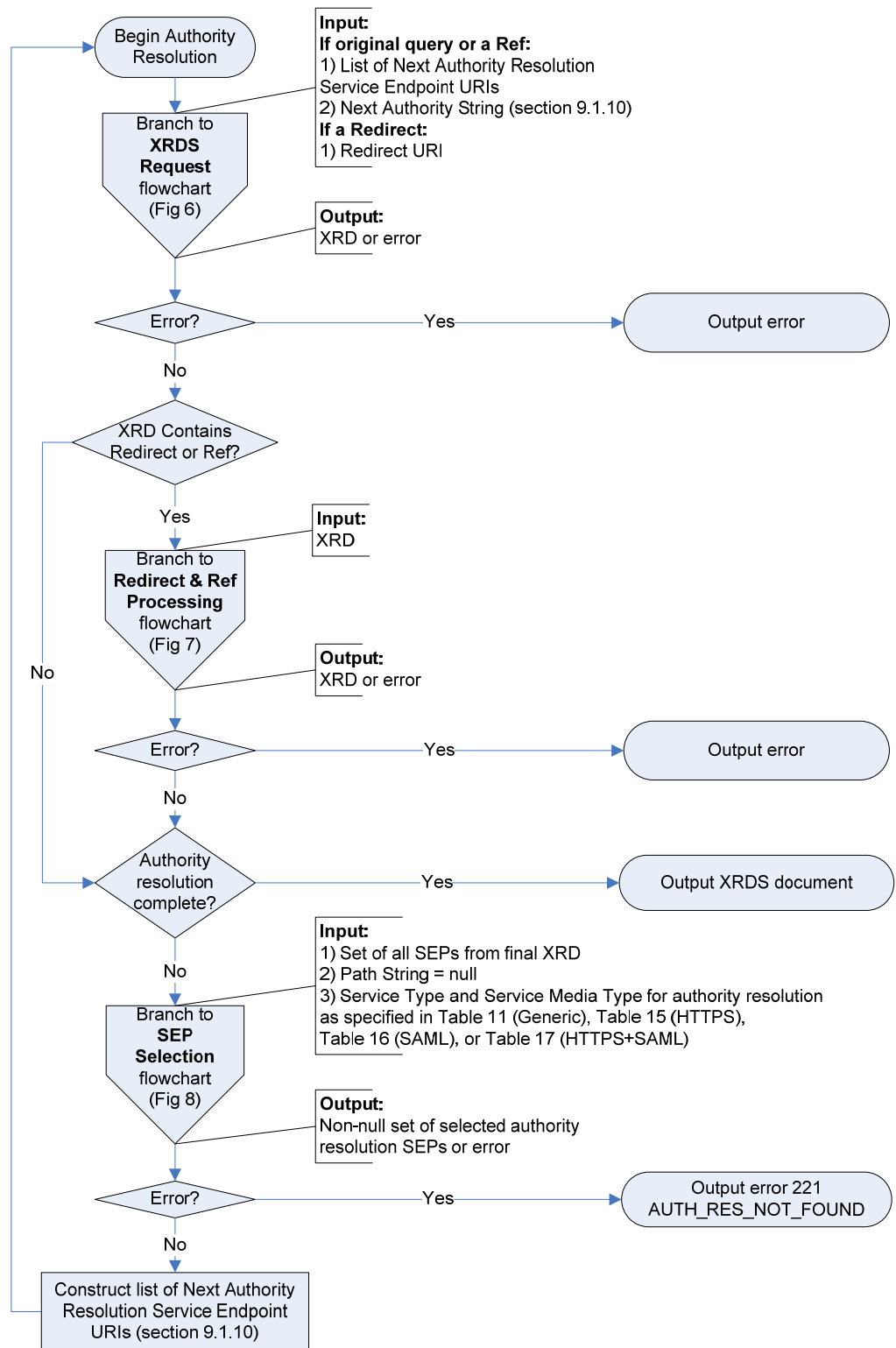
1270   *Figure 5: Authority resolution flowchart.*

1271 Following are the normative requirements for behavior of an XRI resolver and an XRI authority
1272 server when performing generic XRI authority resolution:

1273   1. Each request for an XRDS document using HTTP(S) MUST conform to the requirements
1274   in section 9.1.3.

1275   2. For errors in XRDS document resolution requests, a resolver MUST implement failover
1276   handling as specified in section 9.1.4.

1277   3. The resolver MUST be preconfigured with or have a means of obtaining the XRDS
1278   document describing the community root authority for the XRI to be resolved as defined
1279   in section 9.1.5.

1280   4. The resolver MAY obtain the XRDS document describing the community root authority by
1281   requesting a self-describing XRDS document as defined in section 9.1.6.

1282   5. Resolution of each subsegment in the Authority String after the community root
1283   subsegment MUST proceed in subsegment order (left-to-right) using fully qualified
1284   subsegment values as defined in section 9.1.7.

1285   6. Subsegments that use XRI parenthetical cross-reference syntax MUST be resolved as
1286   defined in section 9.1.8.

1287   7. For each iteration of the authority resolution process, the next authority resolution service
1288   endpoint MUST be selected as specified in section 9.1.9.

1289   8. For each iteration of the authority resolution process, an HTTP(S) URI (called the Next
1290   Authority URI) MUST be constructed according to the algorithm specified in section
1291   9.1.10.

1292   9. A resolver MAY request that a recursing authority server perform resolution of multiple
1293   subsegments as defined in section 9.1.11.

1294   10. For each iteration of the authority resolution process, a resolver MUST perform Redirect
1295   and Ref processing as specified in section 12. Note that if Redirect and Ref processing is
1296   successful, it will result in a nested XRDS document as specified in section 12.5 and
1297   illustrated in Figure 6.

1298

9.1.3 Requesting an XRDS Document using HTTP(S)

1299

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

1301
1302
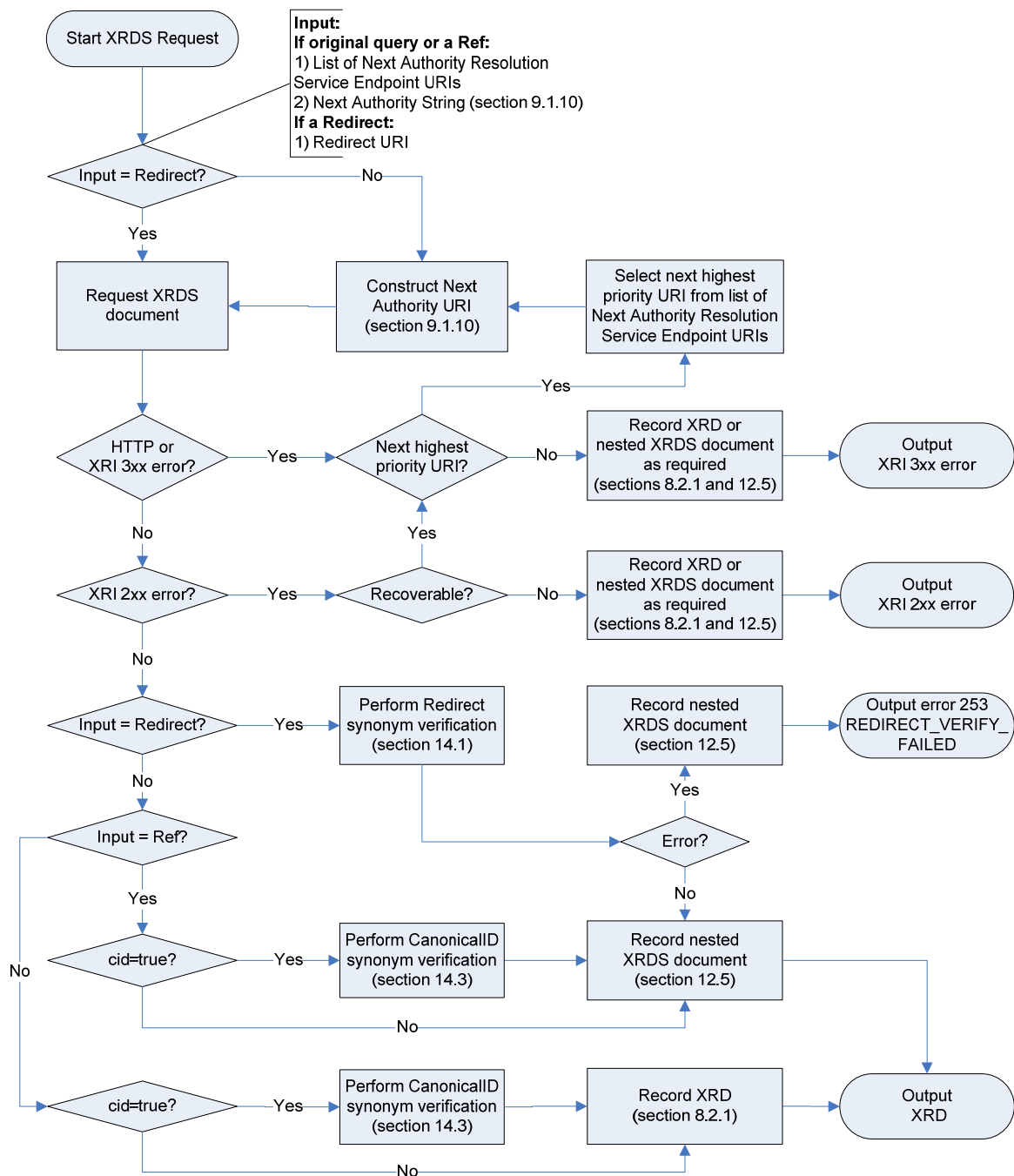
1303 *Figure 6: XRDS request flowchart.*

1304 Note that the term "Record" in Figure 6 means that if the Resolution Output Format is an XRDS
1305 document, this is the logical operation of appending either an XRD or an XRDS document at the
1306 proper nesting level within that output. See the examples in section 12.5.

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

1. Each resolution request MUST be an HTTP(S) GET to the Next Authority URI and MUST contain an Accept header with the media type identifier defined in Table 11. Note that in XRI authority resolution, this Accept header is NOT interpreted as an XRI resolution input parameter, but simply as the media type being requested from the server. This differs from XRI proxy resolution, where the Accept header MAY be used to specify the Service Media Type resolution parameter. See section 11.5.

2. The ultimate HTTP(S) response from an authority server to a successful resolution request MUST contain either: a) a 2XX response with a valid XRDS document containing an XRD element for each authority subsegment resolved, or b) a 304 response signifying that the cached version on the resolver is still valid (depending on the client's HTTP(S) request). There is no restriction on intermediate redirects (i.e., 3XX result codes) or other result codes (e.g., a 100 HTTP response) that eventually result in a 2XX or 304 response through normal operation of [RFC2616].

3. The HTTP(S) response from an authority server MUST return the media type requested by the resolver. The response SHOULD NOT include any subparameters supplied by the resolver in the request. If the resolver receives such parameters in the response, the resolver MUST ignore them and do its own independent verification that the response fulfills the requested parameters.

4. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in the resolution process. In this case, the resolver MUST implement failover handling as specified in section 9.1.4.

5. If all authority resolution service endpoints fail, the resolver SHOULD return the appropriate error code and context message as specified in section 15. In recursing resolution, such an error MUST be returned by the recursing authority server to the resolver as specified in section 15.5.

6. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section 16. In particular, HTTP caching semantics SHOULD be leveraged to the greatest extent possible to maintain the efficiency and scalability of the HTTP-based resolution system. The recommended use of HTTP caching headers is described in more detail in section 16.2.1.

9.1.4 Failover Handling

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

- Critical authority or proxy resolution servers SHOULD be operated from a minimum of two physically separate network locations to prevent a single point of failure.

- Authority or proxy resolution servers handling heavy loads SHOULD operate from multiple servers and take advantage of load balancing technologies.

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

- *DNS round robin/failover*: The domain name of an authority resolution service endpoint URI may be associated with more than one IP address.

- *XRI round robin/failover*: The XRDS document describing an XRI authority may publish multiple URI elements for its authority resolution service endpoint, or multiple authority resolution service endpoints, or both.

1356   To take advantage of both these options, the following rules apply to failover handling:

1357   1. A resolver SHOULD first try an alternate IP address for the current authority resolution
1358   service endpoint if the endpoint uses DNS round robin.

1359   2. If all alternate IP addresses fail, a resolver MUST try the next highest priority authority
1360   resolution URI in the current authority resolution service endpoint, if available.

1361   3. If all URIs in the current authority resolution service endpoint fail, a resolver MUST try the
1362   next highest priority authority resolution service endpoint, if available, until all authority
1363   resolution service endpoints are exhausted.

1364   4. A resolver SHOULD only return an error if all network endpoints associated with the
1365   authority resolution service fail to respond.

1366   IMPORTANT: These rules also apply to any client of an XRI proxy resolver. Failure to observe
1367   this warning means the proxy resolver can become a point of failure.

1368   One final consideration: DNS caching mechanisms should respect the TTL (Time To Live)
1369   settings in DNS records. However, different software languages and frameworks handle DNS
1370   caching differently. It is RECOMMENDED to check the default settings to ensure that a library or
1371   application is not caching DNS results indefinitely.

9.1.5 Community Root Authorities

1373   Identifier management policies are defined on a community-by-community basis. For XRI
1374   identifier authorities, the resolution community is specified by the first (leftmost) subsegment of
1375   the authority component of the XRI. This is referred to as the *community root authority*, and it
1376   represents the authority server(s) that answer resolution queries at this root. When a resolution
1377   community chooses to create a new community root authority, it SHOULD define policies for
1378   assigning and managing identifiers under this authority. Furthermore, it SHOULD define what
1379   resolution protocol(s) may be used for these identifiers.

1380   For an XRI authority, the community root may be either a global context symbol (GCS) character
1381   or top-level cross-reference as specified in section 2.2.1.1 of **[XRISyntax]**. In either case, the
1382   corresponding root XRDS document (or its equivalent) specifies the top-level authority resolution
1383   service endpoints for that community.

1384   The community root authority SHOULD publish a self-describing XRDS document as defined in
1385   section 9.1.6. This XRDS document SHOULD be available at the HTTP(S) URI(s) that serve as
1386   the community's root authority resolution service endpoints. This community root XRDS
1387   document, or its location, must be known *a priori* and is part of the configuration of an XRI
1388   resolver, similar to the specification of root DNS servers for a DNS resolver. Note that it is not
1389   strictly necessary to publish this information in an XRDS document—it may be supplied in any
1390   format that enables configuration of the XRI resolvers in the community. However, publishing a
1391   self-describing XRDS document at a known location simplifies this process and enables dynamic
1392   configuration of community resolvers.

1393   As a best practice, it is RECOMMENDED that community root XRDS document contain:

1394   • The root HTTPS resolution service endpoint(s) if HTTPS trusted resolution is supported.

1395   • A valid self-signed SAML assertion accessible via HTTPS or other secure means if SAML
1396   trusted resolution is supported.

1397   • Both of the above if HTTPS+SAML trusted resolution is supported.

1398   • The service endpoints and supported media types of the community's XRI proxy resolver(s) if
1399   proxy resolution is supported.

1400   For a list of public community root authorities and the locations of their community root XRDS
1401   documents, see the Wikipedia entry on XRI **[WikipediaXRI]**.

9.1.6 Self-Describing XRDS Documents

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

1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution service endpoint, it may use the resolution protocol specified in section 6 to request an XRDS document directly from this HTTP(S) URI. This HTTP(S) URI may be known a priori (as is often the case with community root authorities, above), or it may be discovered from other identifier authorities via the resolution protocols defined in this specification.

2. If the resolver knows: a) an XRI of the target authority as a community root authority, and b) an HTTP(S) URI for a proxy resolver configured for this community root authority, it may use the proxy resolution protocol specifed in section 11 to query the proxy resolver for the community root authority XRI. This query MUST include only a single subsegment identifying the community root authority and MUST NOT include any additional subsegments.

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

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

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

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

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

9.1.7 Qualified Subsegments

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

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

1444

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

1445   *Table 12: Parsing the first subsegment of an XRI that begins with a global context symbol.*

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

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

1447   9.1.8 Cross-References

1448   Any subsegment within an XRI authority component may be a cross-reference (see section 2.2.2
1449   of **[XRISyntax]**). Cross-references are resolved identically to any other subsegment because the
1450   cross-reference is considered opaque, i.e., the value of the cross-reference (including the
1451   parentheses) is the literal value of the subsegment for the purpose of resolution.

1452   Table 14 provides several examples of resolving cross-references. In these examples,
1453   subsegment !b resolves to a Next Authority Resolution Service Endpoint URI of
1454   http://example.com/xri/ and recursing authority resolution is not being requested.
1455

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

1456   *Table 14: Examples of the Next Authority URIs constructed using different types of cross-references.*

1457   9.1.9 Selection of the Next Authority Resolution Service Endpoint

1458   For each iteration of authority resolution, the resolver MUST select the next authority resolution
1459   service endpoint from the current XRD as specified in section 13. For generic authority resolution,
1460   this selection process MUST use the parameters specified in Table 11. For trusted authority
1461   resolution, this selection process MUST use the parameters specified in Table 15, Table 16, or
1462   Table 17. In all cases, an explicit match on the xrd:XRD/xrd:Service/xrd:Type element is
1463   REQUIRED, so during authority resolution, a resolver MUST set the nodefault parameter to a
1464   value of nodefault=type in order to override selection of a default service endpoint as
1465   specified in section 13.3.2.

9.1.10 Construction of the Next Authority URI

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

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

1. Select the highest priority URI of the highest priority authority resolution service endpoint selected in section 9.1.9.

2. Apply the service endpoint URI construction algorithm based the value of the `append` attribute as defined in section 13.7.

3. Append a forward slash ("/") *if the URI does not already end in a forward slash*.

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

- The next fully qualified subsegment to be resolved (see section 9.1.7), or

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

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

BACKWARDS COMPATIBILITY NOTE: Earlier versions of this specification required the Next Authority String to be appended to the *path component* of the Next Authority Resolution Service Endpoint URI. This rule was changed to give XRI authorities greater control over the structure of incoming resolution requests—for example, to enable Next Authority Strings to appear as query parameters.

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

```
if (next-auth-res-sep-uri does not end in "/"):
    append "/" to next-auth-res-sep-uri

if (next-auth-string is not preceded with "*" or "!" delimiter):
    prepend "*" to next-auth-string

append uri-escape(next-auth-string) to next-auth-res-sep-uri
```

9.1.11 Recursing Authority Resolution

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

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

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

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

9.2 IRI Authority Resolution

1516 From the standpoint of generic authority resolution, an IRI authority component represents either
1517 a DNS name or an IP address at which an XRDS document describing the authority may be
1518 retrieved using HTTP(S). Thus IRI authority resolution simply involves making an HTTP(S) GET
1519 request to a URI constructed from the IRI authority component. The resulting XRDS document
1520 can then be consumed in the same manner as one obtained using XRI authority resolution.

1521 While the use of IRI authorities provides backwards compatibility with the large installed base of
1522 DNS- and IP-identifiable resources, IRI authorities do not support the additional layer of
1523 abstraction, delegation, and extensibility offered by XRI authority syntax. Therefore IRI authorities
1524 are NOT RECOMMENDED for new deployments of XRI identifiers.

1525 This section defines IRI authority resolution as a simple extension to the XRI authority resolution
1526 protocol defined in the preceding section.

9.2.1 Service Type and Media Type

1528 Because IRI authority resolution takes place at a level "below" XRI authority resolution, it cannot
1529 be described in an XRD, and thus there is no corresponding resolution service type. IRI authority
1530 resolution uses the same media type as generic XRI authority resolution.

9.2.2 Protocol

1532 Following are the normative requirements for IRI authority resolution that differ from generic XRI
1533 authority resolution:

1534 1. The Next Authority URI (section 9.1.10) is constructed by extracting the entire IRI
1535   authority component and prepending the string `http://`. See the exception in section
1536   9.2.3.

1537 2. The HTTP GET request MUST include an HTTP Accept header containing only the
1538   following:

1539   `Accept: application/xrds+xml`

1540 3. The HTTP GET request MUST have a `Host:` header (as defined in section 14.23 of
1541   **[RFC2616]**) containing the value of the IRI authority component. For example:

1542   `Host: example.com`

1543 4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document
1544   containing the XRD describing that authority.

1545 5. The responding server MUST use the value of the `Host:` header to populate the
1546   `xrd:XRD/xrd:Query` element in the resulting XRD.

1547 Note that because IRI authority resolution is required to process the entire IRI authority
1548 component in a single step, recursing authority resolution does not apply.

9.2.3 Optional Use of HTTPS

1550 Section 10 of this specification defines trusted resolution only for XRI authorities. Trusted
1551 resolution is not defined for IRI Authorities. If, however, an IRI authority is known to respond to
1552 HTTPS requests (by some means outside the scope of this specification), then the resolver MAY
1553 use HTTPS as the access protocol for retrieving the authority's XRD. If the resolver is satisfied,

1554   via transport level security mechanisms, that the response is from the expected IRI authority, the
1555   resolver MAY consider this an HTTPS trusted resolution response as defined in section 10.1.

10 Trusted Authority Resolution Service

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.

10.1 HTTPS

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.

10.1.1 Service Type and Service Media Type

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

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

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.

10.1.2 Protocol

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

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.

10.2 SAML

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

1593 HTTPS trusted resolution as defined in section 10.3. Message confidentiality may also be
1594 achieved with other security protocols used in conjunction with this specification. SAML trusted
1595 resolution also does not provide a means for an authority to provide different responses for
1596 different requestors; client authentication is explicitly out-of-scope for version 2.0 of XRI
1597 resolution.

1598 10.2.1 Service Type and Service Media Type

1599 The protocol defined in this section is identified by the values in Table 16.

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

1600 *Table 16: Service Type and Service Media Type values for SAML trusted authority resolution.*

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

1605 10.2.2 Protocol

1606 10.2.2.1 Client Requirements

1607 For a resolver, trusted resolution is identical to the generic resolution protocol (section 9.1) with
1608 the addition of the following requirements:

1609   1. All authority resolution service endpoints MUST be selected using the values defined in
1610   Table 16. A resolver SHOULD NOT request SAML trusted resolution service from an
1611   authority unless the authority advertises a resolution service endpoint matching these
1612   values.

1613   2. Authority resolution requests MAY use either the HTTP or HTTPS protocol. The latter is
1614   RECOMMENDED for confidentiality.

1615   3. All authority resolution requests MUST contain an HTTP(S) Accept header with the
1616   media type identifier defined in Table 16 (including the `saml=true` subparameter). This
1617   is the media type of the requested response.

1618 IMPORTANT: Clients willing to accept either generic or trusted responses MAY use a
1619 combination of media type identifiers in the Accept header as described in section 14.1 of
1620 [RFC2616]. Media type identifiers SHOULD be ordered according to the client's preference for
1621 the media type of the response. If a client performing generic authority resolution receives an
1622 XRD containing SAML elements, it MAY choose not to validate the signature or perform any
1623 processing of these elements.

1624   4. A resolver MAY request recursing authority resolution of multiple subsegments as
1625   defined in section 10.2.3.

1626   5. The resolver MUST individually validate each XRD it receives in the resolution chain
1627   according to the rules defined in section 10.2.4. When `xrd:XRD` elements come both
1628   from freshly-retrieved XRDS documents and from a local cache, a resolver MUST ensure
1629   that these requirements are satisfied each time a resolution request is performed.

10.2.2.2 Server Requirements

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

1. The HTTP(S) response to a trusted resolution request MUST include a content type of `application/xrds+xml;saml=true`.

2. The XRDS document returned by the resolution service MUST contain a `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid per the processing rules described by [SAML].

3. The `saml:Assertion` element MUST contain a valid enveloped digital signature as defined by [XMLDSig] and as constrained by section 5.4 of [SAML].

4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML assertion. Specifically, the signature MUST contain a single `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML assertion. The URI reference MUST NOT be empty and it MUST refer to the identifier contained in the `xrd:XRD/@xml:id` attribute.

5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain a `ds:KeyInfo` element. If this element is included, it MUST describe the key used to verify the digital signature element. However, because the signing key is known in advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the `ds:Signature` element of the SAML assertion.

6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST match the XRI authority subsegment requested by the client.

7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD advertising availability of trusted resolution service from this authority as required in section 10.2.5.

8. The `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be present and equal to the `xrd:XRD/xrd:Query` element.

9. The `NameQualifier` attribute of the `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be present and MUST be equal to the `xrd:XRD/xrd:ProviderID` element.

10. There MUST be exactly one `saml:AttributeStatement` present in the `xrd:XRD/saml:Assertion` element. It MUST contain exactly one `saml:Attribute` element with a `Name` attribute value of `xri://$xrd*($v*2.0)`. This `saml:Attribute` element MUST contain exactly one `saml:AttributeValue` element whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent of the `saml:Assertion` element.

10.2.3 Recursing Authority Resolution

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

10.2.4 Client Validation of XRDs

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

1. The `xrd:XRD/saml:Assertion` element MUST be present.

2. This assertion MUST be valid per the processing rules described by [SAML].

3. The `saml:Assertion` MUST contain a valid enveloped digital signature as defined by [XMLDSig] and constrained by Section 5.4 of [SAML].

4. The signature MUST apply to the `xrd:XRD` element containing the signed SAML assertion. Specifically, the signature MUST contain a single `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference MUST refer to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent of the signed SAML assertion.

5. If the digital signature enveloped by the SAML assertion contains a `ds:KeyInfo` element, the resolver MAY reject the signature if this key does not match the signer's expected key as specified by the `ds:KeyInfo` element present in the XRD Descriptor that was used to describe the current authority. See section 10.2.5.

6. The value of the `xrd:XRD/xrd:Query` element MUST match the subsegment whose resolution resulted in the current XRD.

7. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in any XRD advertising availability of trusted resolution service from this authority as required in section 10.2.5.

8. The value of the `xrd:XRD/xrd:ProviderID` element MUST match the value of the `NameQualifier` attribute of the `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.

9. The value of the `xrd:XRD/xrd:Query` element MUST match the value of the `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element.

10. There MUST exist exactly one `xrd:XRD/saml:Assertion/saml:AttributeStatment` with exactly one `saml:Attribute` element that has a `Name` attribute value of `xri://$xrd*($v*2.0)`. This `saml:Attribute` element must have exactly one `saml:AttributeValue` element whose text value is a URI reference to the `xml:id` attribute of the `xrd:XRD` element that is the immediate parent of the signed SAML assertion.

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

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

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

10.2.5 Correlation of ProviderID and KeyInfo Elements

1723 Each XRI authority participating in SAML trusted authority resolution MUST be associated with at
1724 least one unique persistent identifier expressed in the
1725 `xrd:XRD/xrd:Service/xrd:ProviderID` element of any XRD advertising trusted authority
1726 resolution service. This ProviderID value MUST NOT ever be reassigned to another XRI
1727 authority. While a ProviderID may be any valid URI that meets these requirements, it is
1728 STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI
1729 **[XRISyntax]** or a URN **[RFC2141]**.

1730 The purpose of ProviderIDs in XRI resolution is to enable resolvers to correlate the metadata in
1731 an XRD advertising SAML trusted authority resolution service with the response received from a
1732 SAML trusted resolution service endpoint. If the signed XRD response contains the same
1733 ProviderID as the XRD used to advertise a service, and the resolver has reason to trust the
1734 signature, the resolver can trust that the XRD response has not been maliciously replaced with
1735 another XRD.

1736 There is no defined discovery process for the ProviderID for a community root authority; it must
1737 be published in a self-describing XRDS document (or other equivalent description—see sections
1738 9.1.5 and 9.1.6) and verified independently. Once the community root XRDS document is known,
1739 the ProviderID for delegated XRI authorities within this community MAY be discovered using the
1740 `xrd:XRD/xrd:Service/xrd:ProviderID` element of authority resolution service endpoints.
1741 This trust mechanism MAY also be used for other services offered by an authority.

1742 In addition, the metadata necessary for SAML trusted authority resolution or other SAML **[SAML]**
1743 interactions MAY be discovered using the `ds:KeyInfo` element (section 4.2.) Again, if this
1744 element is present in an XRD advertising SAML authority resolution service (or any other
1745 service), and the client has reason to trust this XRD, the client MAY use the associated
1746 ProviderID to correlate the contents of this element with a signed response.

1747 To assist resolvers in using this key discovery mechanism, it is important that trusted authority
1748 servers be configured to sign responses in such a way that the signature can be verified using the
1749 correlated `ds:KeyInfo` element. For more information, see **[SAML]**.

1750 ## 10.3 HTTPS+SAML

1751 ## 10.3.1 Service Type and Service Media Type

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

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

1753 *Table 17: Service Type and Service Media Type values for HTTPS+SAML trusted authority resolution.*

1754 An HTTPS+SAML trusted resolution service endpoint advertised in an XRDS document MUST
1755 use the Service Type identifier and Service Media Type identifier defined in Table 17 (including
1756 the `https=true` and `saml=true` subparameters). In addition, the identifier authority MUST use
1757 an HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

1758

10.3.2 Protocol

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

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

2. All authority resolution requests and responses, including the starting request to a community root authority, MUST conform to both the requirements of the HTTPS trusted resolution protocol defined in section 10.1 and the SAML trusted resolution protocol defined in section 10.2.

3. All authority resolution requests MUST contain an HTTPS Accept header with the media type identifier defined in Table 17 (including both the `https=true` and `saml=true` parameters). This MUST be interpreted as the value of the Resolution Output Format input parameter.

4. If the resolver finds that an authority in the resolution chain does not support both HTTPS and SAML, the resolver MUST return a 23x error as defined in section 15.

11 Proxy Resolution Service

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

Proxy resolution is useful for:

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

- Optimizing XRD caching for a resolution community (a *caching proxy resolver*). Proxy resolvers SHOULD use caching to resolve the same QXRIs or QXRI components for multiple clients as defined in section 16.4.

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

11.1 Service Type and Media Types

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

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

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

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

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

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

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

11.2 HXRIs

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

1810 • It allows XRIs to be used anyplace an HTTP URI can appear, including in Web pages,
1811   electronic documents, email messages, instant messages, etc.

1812 • It allows XRI-aware processors and search agents to recognize an HXRI and extract the
1813   embedded XRI for direct resolution, processing, and indexing.

1814 To make this syntax as simple as possible for XRI-aware processors or search agents to
1815 recognize, an HXRI consists of a fully qualified HTTP or HTTPS URI authority component that
1816 begins with the domain name segment "`xri.`". The QXRI is then appended as the entire local
1817 path (and query component, if present). The QXRI MUST NOT include the `xri://` prefix and
1818 MUST be in URI-normal form as defined in **[XRISyntax]**. (If a proxy resolver receives an HXRI
1819 containing a QXRI beginning with an `xri://` prefix, it SHOULD remove it before continuing.) In
1820 essence, the proxy resolver URI (including the forward slash after the domain name) serves as a
1821 machine-readable alternate prefix for an absolute XRI in URI-normal form.

1822 The normative ABNF for an HXRI is defined below based on the `ireg-name`, `xri-hier-part,`
1823 and `iquery` productions defined in **[XRISyntax]**. XRIs that need to be understood by non-XRI-
1824 aware clients SHOULD be published as HTTP URIs conforming to this HXRI production.

```
1825   HXRI             = proxy-resolver "/" QXRI

1826   proxy-resolver   = ( "http://" / "https://" ) proxy-reg-name

1827   proxy-reg-name   = "xri." ireg-name

1828   QXRI             = xri-hier-part [ "?" i-query ]
```

1829 URI processors that recognize XRIs SHOULD interpret the local part of an HTTP or HTTPS URI
1830 (the path segment(s) and optional query segment) as an XRI provided that: a) it conforms to this
1831 ABNF, and b) the first segment of the path conforms to the xri-authority or iauthority productions
1832 in **[XRISyntax]**.

1833 For references to communities that offer public XRI proxy resolution services, see the Wikipedia
1834 entry on XRI **[WikipediaXRI]**.

11.3 HXRI Query Parameters

1836 In proxy resolution, the XRI resolution input parameters defined in section 8.1 are bound to an
1837 HTTP(S) interface using the conventional web model of encoding them in an HTTP(S) URI, which
1838 in this case is an HXRI. The binding of the logical parameter names to HXRI component parts is
1839 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

1840 *Table 19: Binding of logical XRI resolution parameters to QXRI query parameters.*

1841

1842   Following are the rules for the use of the parameters specified in Table 19.

1843   1. The QXRI MUST be normalized as specified in section 11.2.

1844   2. If the original QXRI has an existing query component, the HXRI query parameters MUST
1845   be appended to that query component.

1846   IMPORTANT: The query parameter names in Table 19 were chosen to minimize the probability of
1847   collision with any pre-existing query parameter names in the QXRI. If there is any conflict, the
1848   pre-existing query parameter names MUST be percent-encoded prior to transformation into an
1849   HXRI.

1850   3. After proxy resolution, the HXRI query parameters MUST subsequently be removed from
1851   the QXRI query component. The existing QXRI query component MUST NOT be altered
1852   in any other way, i.e., it must be passed through with no changes in parameter order,
1853   escape encoding, etc.

1854   4. If the original QXRI does not have a query component, one MUST be added to pass any
1855   HXRI query parameters. After proxy resolution, this query component MUST be entirely
1856   removed.

1857   5. If the original QXRI had a null query component (only a leading question mark), or a
1858   query component consisting of only question marks, *one additional leading question mark*
1859   MUST be added before adding any HXRI query parameters. After proxy resolution, any
1860   HXRI query parameters and exactly one leading question mark MUST be removed. See
1861   the URI construction steps defined in section 13.6.

1862   6. Each HXRI query parameter MUST be delimited from other parameters by an ampersand
1863   ("&").

1864   7. Each HXRI query parameter MUST be delimited from its value by an equals sign ("=").

1865   8. If an HXRI query parameter includes one of the media type parameters defined in Table
1866   6, it MUST be delimited from the HXRI query parameter with a semicolon (";").

1867   9. The fully-composed HXRI MUST be encoded and decoded as specified in section 11.4.

1868   10. If any HXRI query parameter name is included but its value is empty, the value of the
1869   parameter MUST be considered null.

11.4 HXRI Encoding/Decoding Rules

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

1878   Before any HXRI-specific encoding steps are performed, the QXRI portion of the HXRI (including
1879   all HXRI query parameters) MUST be transformed into URI-normal form as defined in section 2.3
1880   of **[XRISyntax]**. This means characters not allowed in URIs, such as SPACE, or characters that
1881   are valid only in IRIs, such as UCS characters above the ASCII range, MUST be percent
1882   encoded. Also, the plus sign character ("+") MUST NOT be used to encode the SPACE character
1883   because in decoding the percent-encoded sequence %2B MUST be interpreted as the plus sign
1884   character ("+").

1885   Once the HXRI is in URI-normal form, the following sequence of encoding steps MUST be
1886   performed in the order specified before an HXRI is submitted to a proxy resolver.

1887   IMPORTANT: this sequence of steps is not idempotent, so it MUST be performed only once.

1888   1. First, in order to preserve percent-encoding when the HXRI is passed through a web
1889   server, all percent signs MUST be themselves percent-encoded, i.e., a SPACE encoded
1890   as `%20` will become `%2520`.

1891   2. Second, to prevent misinterpretation of HXRI query parameters, any occurrences of the
1892   ampersand character ("`&`") within an HXRI query parameter that are NOT used to delimit
1893   it from another query parameter MUST be percent encoded using the sequence `%26`.

1894   3. Third, to prevent misinterpretation of the semicolon character by the web server, any
1895   semicolon used to delimit one of the media type parameters defined in Table 6 from the
1896   media type value MUST be percent-encoded using the sequence `%3B`.

1897 To decode an encoded HXRI back into URI-normal form, the above sequence of steps MUST be
1898 performed in reverse order. Again, the sequence is not idempotent so it MUST be performed only
1899 once.

1900 Table 20 illustrates the components of an example HXRI before transformation to URI-normal
1901 form. The characters requiring percent encoding are highlighted in **red**. Note the space in the
1902 string `hello planète`. Also, for purposes of illustration, the Type component contains a query
1903 string (which would not normally appear in a Type identifier).

QXRI	`https://xri.example.com/=example*r`**é**`sum`**é**`/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 plan`**è**`te`
_xrd_m	`_xrd_m=application/atom+xml`

1904   *Table 20: Example of HXRI components prior to transformation to URI-normal form.*

1905 Table 21 illustrates these components after transformation to URI-normal form. Characters that
1906 have been percent-encoded are in **blue**. Characters still requiring percent encoding according to
1907 the rules defined in this section are highlighted in **red**.

QXRI	`https://xri.example.com/=example*r`**%E9**`sum`**%E9**`/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`**%20**`plan`**%E8**`te`
_xrd_m	`_xrd_m=application/atom+xml`

1908   *Table 21: Example of HXRI components after transformation to URI-normal form.*

1909 Table 22 illustrates the components after all encoding rules defined in this section are applied.

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

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

1911

1912   Following is the fully-encoded HXRI:

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

1917   Following is the fully decoded HXRI returned to URI-normal form. Note that the proxy resolver
1918   MUST leave the HXRI in URI-normal form for any further processing.

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

11.5 HTTP(S) Accept Headers

1924   In proxy resolution, one XRI resolution input parameter, the Service Media Type (section 8.1.4)
1925   MAY be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. The
1926   following rules apply to this input:

1927   1. As described in section 14.1 of **[RFC2616]**, the Accept header content type MAY consist
1928   of multiple media type identifiers. If so, the proxy resolver MUST choose only one to
1929   accept. A proxy resolver client SHOULD order media type identifiers according to the
1930   client's preference and a proxy resolver server SHOULD choose the client's highest
1931   preference.

1932   2. If the value of the Accept header content type is null, this MUST be interpreted as the
1933   value of the Service Media Type parameter.

1934   3. If the value of the Service Media Type parameter is explicitly set via the _xrd_m query
1935   parameter in the HXRI (including to a null value), this MUST take precedence over any
1936   value set via an HTTP(S) Accept header.

11.6 Null Resolution Output Format

1938   Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a
1939   resolution request where the Resolution Output Format input parameter value is null—either
1940   because this parameter is absent or because it was explicitly set to null using the _xrd_r query
1941   parameter.

1942   If the value of the Resolution Output Format value is null, a resolver MUST proceed as if the
1943   following media type parameters had the following values: https=false, saml=false,
1944   refs=true, sep=true, nodefault_t=false, nodefault_p=false,
1945   nodefault_m=false, and uric=false. In addition, the output MUST be an HTTP(S) redirect
1946   as defined in the following section.

11.7 Outputs and HTTP(S) Redirects

1948   For all values of the Resolution Output Format parameter except null, a proxy resolver MUST
1949   follow the output rules defined in section 8.2.

1950   If the value of the Resolution Output Format is null, and the output is not an error, a proxy
1951   resolver MUST follow the rules for output of a URI List as defined in section 8.2.3. However,
1952   instead of returning a URI list, it MUST return the highest priority URI (the first one in the list) as
1953   an HTTP(S) 3XX redirect with the Accept header content type set to the value of the Service
1954   Media Type parameter.

1955   If the output is an error, a proxy resolver SHOULD return a human-readable error message as
1956   specified in section 15.4.

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

1961 11.8 Differences Between Proxy Resolution Servers

1962 An XRI proxy resolution request MAY be sent to any proxy resolver that will accept it. All XRI
1963 proxy resolvers SHOULD deliver uniform responses given the same QXRI and other input
1964 parameters. However, because proxy resolvers may potentially need to make decisions about
1965 network errors, Redirect and Ref processing, and trust policies on behalf of the client they are
1966 proxying, and these decisions may be based on local policy, in some cases different proxy
1967 resolvers may return different results.

1968 11.9 Combining Authority and Proxy Resolution Servers

1969 The majority of DNS nameservers are recursing nameservers that answer both queries for which
1970 they are authoritative and queries which they must forward to other nameservers. The same rule
1971 applies in XRI architecture: in many cases the optimum configuration will be combining an
1972 authority server and proxy resolver in the same server. This server can publish a self-describing
1973 XRDS document (section 9.1.6) that advertises both its authority resolution and proxy resolution
1974 service endpoints. It can also optimize caching of XRDs for clients in its resolution community
1975 (see section 16.4).

12 Redirect and Ref Processing

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

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

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

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

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

Table 23: Comparison of Redirect and Ref elements.

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

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

1998   Figure 7 (non-normative) illustrates the logical flow of Redirect and Ref processing.
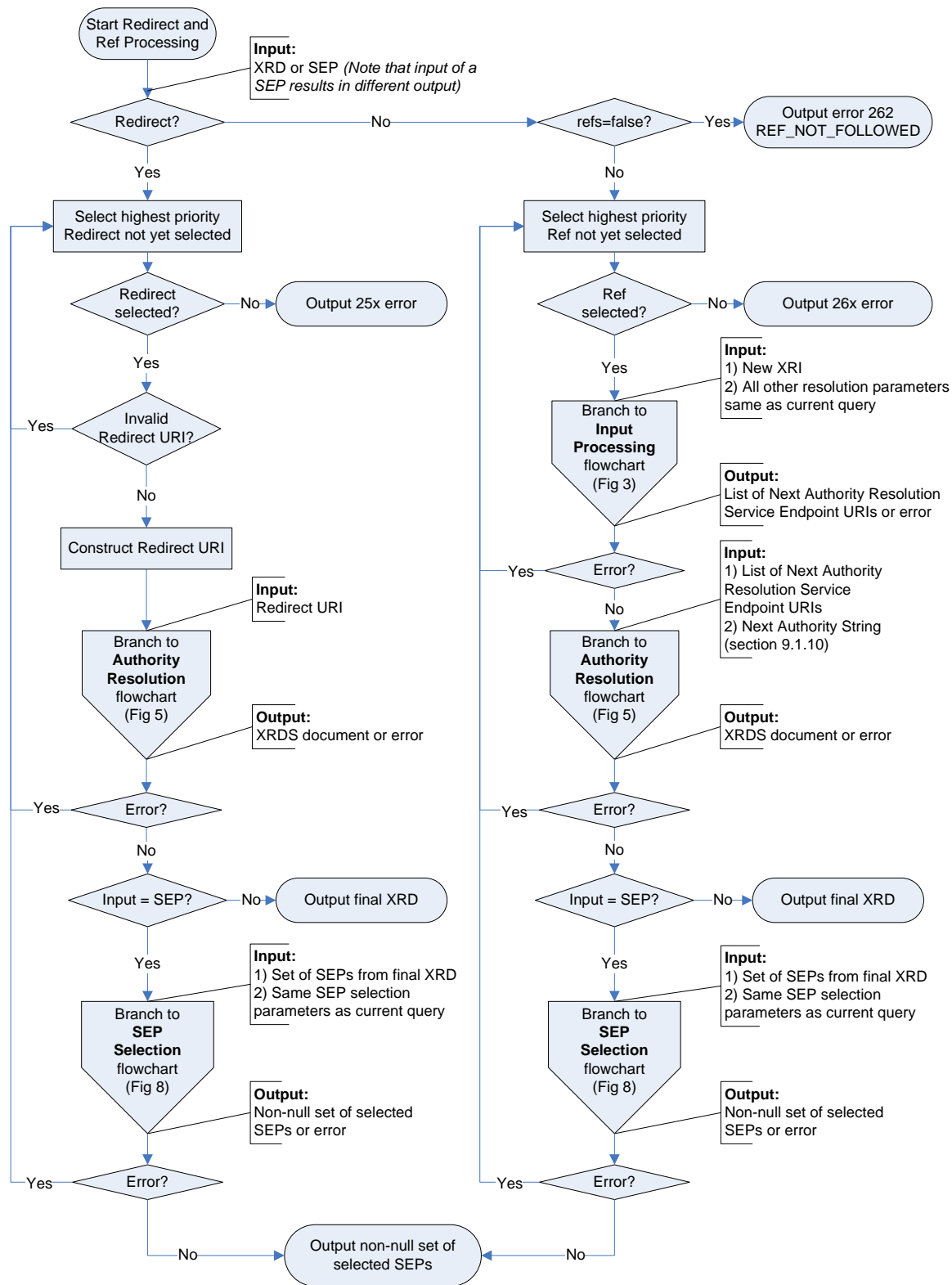
1999
2000

2001   *Figure 7: Redirect and Ref processing flowchart.*

12.1 Cardinality

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

- At the XRD level, an XRD MAY contain only one of the following: zero-or-more `xrd:Redirect` or zero-or-more `xrd:Ref` elements.

- At the SEP level, a SEP MAY contain only one of the following: zero-or-more `xrd:URI` elements, zero-or-more `xrd:Redirect` elements, or zero-or-more `xrd:Ref` elements.

12.2 Precedence

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

1. If a Redirect or Ref element is present at the XRD level, it MUST be processed immediately before a resolver continues with authority resolution, performs service endpoint selection (required or optional), or returns its final output. This rule applies recursively to all XRDS documents resolved as a result of Redirect or Ref processing.

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

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

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

1. XRDS authors SHOULD NOT put any service endpoints in an XRD that contains a Redirect or Ref at the XRD level because by definition these service endpoints will be ignored.

2. XRDS authors SHOULD use a Redirect or Ref element at the XRD level if they wish to relocate or delegate resolution behavior regardless of any service endpoint query.

3. XRDS authors SHOULD use a Redirect or Ref element in a service endpoint for which they expect a POSITIVE match as defined in section 13.4.1 if they wish to control resolution behavior based an explicit service endpoint match.

4. XRDS authors SHOULD use a Redirect or Ref element in a service endpoint for which they expect a DEFAULT match as defined in section 13.4.1 if they wish to control resolution behavior based on the absence of an explicit service endpoint match.

5. XRDS authors SHOULD NOT include two or more SEPs of equal priority in an XRD if they contain Redirects or Refs that will make resolution ambiguous or non-deterministic.

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

12.3 Redirect Processing

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

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

1. To process a Redirect at either the XRD or SEP level, the resolver MUST begin by selecting the highest priority `xrd:XRD/xrd:Redirect` element in the XRD or SEP.

2. If the value of the resolution subparameter `https` is FALSE, or the subparameter is absent or empty, the value of the selected `xrd:Redirect` element MUST be EITHER a valid HTTP URI or a valid HTTPS URI. If not, the resolver MUST select the next highest priority `xrd:Redirect` element. If all instances of this element fail, the resolver MUST stop and return the error `251 INVALID_REDIRECT` in the XRD containing the Redirect or as a plain text error message as specified in section 15.

3. If the value of the resolution subparameter `https` is TRUE, the value of the selected `xrd:Redirect` element MUST be a valid HTTPS URI. If not, the resolver MUST select the next highest priority `xrd:Redirect` element. If all instances of this element fail, the resolver MUST stop and return the error `252 INVALID_HTTPS_REDIRECT` in the XRD containing the Redirect or as a plain text error message as specified in section 15.

4. Once a valid `xrd:Redirect` element has been selected, if the `xrd:XRD/xrd:Redirect` element includes the `append` attribute, the resolver MUST construct the final HTTP(S) URI as defined in section 13.7.

5. The resolver MUST request a new XRDS document from the final HTTP(S) URI using the protocol defined in section 9.1.3. If the Resolution Output Format is an XRDS document, the resolver MUST embed a nested XRDS document containing an XRD representing the Redirect as specified in section 12.5.

6. If resolution of an `xrd:Redirect` element fails during the authority resolution phase of the original resolution query, or if resolution of an `xrd:Redirect` element fails during the optional service endpoint selection phase OR the final XRD does not contain the requested SEP, then the resolver MUST report the error in the final XRD of the nested XRDS document using the status codes defined in section 15. (One nested XRDS document will be added for each Redirect attempted by the resolver.) The resolver MUST then select the next highest priority `xrd:Redirect` element from the original XRD or SEP and repeat rule 7. For more details, see section 12.6, *Recursion and Backtracking*.

7. If resolution of all `xrd:Redirect` elements in the XRD or SEP that originally triggered Redirect processing fails, the resolver MUST stop and return a 25x error in the XRD containing the Redirect or as a plain text error message as specified in section 15. The resolver MUST NOT try any other SEPs even if multiple SEPs were selected as specified in section 13.

8. If resolution succeeds, the resolver MUST verify the synonym elements in the new XRD as specified in section 14.1. If synonym verification fails, the resolver MUST stop and return the error specified in that section.

9. If the value of the resolution subparameter `saml` is TRUE, the resolver MUST verify the signature on the XRD as specified in section 10.2.4. If signature verification fails, the resolver MUST stop and return the error specified in that section.

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

12.4 Ref Processing

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

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

1. Ref processing is only performed if the value of the `refs` subparameter (Table 6) is TRUE or it is absent or empty. If the value is FALSE and the XRD contains at least one `xrd:Ref` element that could be followed to complete the resolution query, the resolver MUST stop and return the error `262 REF_NOT_FOLLOWED` in the XRD containing the Ref or as a plain text error message as defined in section 15. The rules below presume that `refs=true`.

2. To process a Ref at either the XRD or SEP level, the resolver MUST begin by selecting the highest priority `xrd:XRD/xrd:Ref` element from the XRD or SEP.

3. The value of the selected `xrd:Ref` element MUST be a valid absolute XRI. If not, the resolver MUST select the next highest priority `xrd:Ref` element. If all instances of this element fail, the resolver MUST stop and return the error `261 INVALID_REF` in the XRD containing the Ref or as a plain text error message as defined in section 15.

4. Once a valid `xrd:XRD/xrd:Ref` value is selected, the resolver MUST begin resolution of a new XRDS document from this XRI using the protocols defined in this specification. Other than the QXRI, the resolver MUST use the same resolution query parameters as the original query. If the Resolution Output Format is an XRDS document, the resolver MUST embed a nested XRDS document containing an XRD representing the Ref as defined in section 12.5.

5. If resolution of an `xrd:Ref` element fails during the authority resolution phase of the original resolution query, or if resolution of an `xrd:Ref` element fails during the optional service endpoint selection phase OR the final XRD does not contain the requested service endpoint, then the resolver MUST record the nested XRDS document as far as resolution was successful, including the relevant status codes for each XRD as specified in section 15. The resolver MUST then select the next highest priority `xrd:Ref` element as specified above and repeat rule 5. For more details, see section 12.6, *Recursion and Backtracking*.

6. If resolution of all `xrd:Ref` elements in the XRD or SEP originating Ref processing fails, the resolver MUST stop and return a 26x error in the XRD containing the Ref or as a plain text error message as specified in section 15. The resolver MUST NOT try any other SEPs even if multiple SEPs were selected as specified in section 13.

7. If resolution of an `xrd:Ref` element succeeds and `cid=true`, the resolver MUST perform CanonicalID verification across all XRDs in the nested XRDS document as specified in section 14.3. Note that each set of XRDs in each new nested XRDS document produced as a result of Redirect or Ref processing constitutes its own CanonicalID verification chain. *CanonicalID verification never crosses between XRDS documents.* See section 12.5 for examples.

8. If resolution of an `xrd:Ref` element succeeds and the final XRD contains the service endpoint(s) necessary to continue or complete the original resolution query, further authority resolution or service endpoint selection MUST continue based on the final XRD.

12.5 Nested XRDS Documents

2143

2144 Processing of a Redirect or Ref ALWAYS produces a new XRDS document that describes the
2145 Redirect or Ref that was followed, even if the result was an error. If the final requested Resolution
2146 Output Format is NOT an XRDS document, this new XRDS document is only needed to obtain
2147 the metadata necessary to continue or complete resolution. However, if the final requested
2148 Resolution Output Format is an XRDS document, each XRDS document produced as a result of
2149 Redirect or Ref processing MUST be nested inside the outer XRDS document immediately
2150 following the `xrd:XRD` element containing the `xrd:Redirect` or `xrd:Ref` element being
2151 followed. If more than one Redirect or Ref element is resolved due to an error, the corresponding
2152 nested XRDS documents MUST be included in the same order as the Redirect or Ref elements
2153 that were followed to produce them.

2154 Each new XRDS document is a recursive authority resolution call and MUST conform to all
2155 authority resolution requirements. In addition, the following rules apply:

2156 • For a Redirect, the `xrds:XRDS/@redirect` attribute of the nested XRDS document MUST
2157   contain the fully-constructed HTTP(S) URI it describes as specified in section 12.3.

2158 • For a Ref, the `xrds:XRDS/@ref` attribute of the nested XRDS document MUST contain the
2159   exact value of the `xrd:XRD/xrd:Ref` element it describes.

2160 This allows a consuming application to verify the complete chain of XRDs obtained to resolve the
2161 original query identifier even if resolution traverses multiple Redirects or Refs, and even if errors
2162 were encountered. Like the outer XRDS document, nested XRDS documents MUST NOT include
2163 an XRD for the community root subsegment because this is part of the configuration of the
2164 resolver.

2165 In addition, during SAML trusted resolution, if a nested XRDS document includes an XRD with an
2166 `xml:id` attribute value matching the `xml:id` attribute value of any previous XRD in the chain of
2167 resolution requests beginning with the original QXRI, the resolver MUST replace this XRD with an
2168 empty XRD element. The resolver MUST set this empty element's `idref` attribute value to the
2169 value of the `xml:id` attribute of the matched XRD element. This prevents conflicting `xml:id`
2170 values.

12.5.1 Redirect Examples

2171

Example #1:

2172

2173 In this example the original query identifier is `xri://@a`. The first XRD contains an XRD-level
2174 Redirect to `http://a.example.com/`. The elements and attributes specific to Redirect
2175 processing are shown in **bold**. CanonicalIDs are included to illustrate the synonym verification
2176 rule in section 12.3.

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

```
2193            </Service>
2194          </XRD>
2195        </XRDS>
2196     </XRDS>
```

Example #2:

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

```
2203     <XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
2204       <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2205         <Query>*a</Query>
2206         <ProviderID>xri://@</ProviderID>
2207         <CanonicalID>xri://@!1</CanonicalID>  ;XRDS #1 CID #1
2208         ...
2209         <Service>
2210           <Type>xri://$res*auth*($v*2.0)</Type>
2211           <URI>http://a.example.com/</URI>
2212         </Service>
2213       </XRD>
2214       <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2215         <Query>*b</Query>
2216         <ProviderID>xri://@!1</ProviderID>
2217         <CanonicalID>xri://@!1!2</CanonicalID>  ;XRDS #1 CID #2
2218         ...
2219         <Service>
2220           <Type>xri://$res*auth*($v*2.0)</Type>
2221           <Redirect>http://other.example.com</Redirect>
2222         </Service>
2223       </XRD>
2224       <XRDS redirect="http://other.example.com">
2225         <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2226           <Query>*b</Query>
2227           <ProviderID>xri://@!1</ProviderID>
2228           <CanonicalID>xri://@!1!2</CanonicalID>  ;SAME AS XRDS #1 CID #2
2229           ...
2230           <Service>
2231             <Type>xri://$res*auth*($v*2.0)</Type>
2232             <URI>http://b.example.com/</URI>
2233           </Service>
2234         </XRD>
2235       </XRDS>
2236       <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2237         <Query>*c</Query>
2238         <ProviderID>xri://@!1!2</ProviderID>
2239         <CanonicalID>xri://@!1!2!3</CanonicalID>  ;XRDS #1 CID #3
2240         ...
2241         <Service>
2242         ...final service endpoints described here...
2243         </Service>
2244       </XRD>
2245     </XRDS>
2246
2247
```

Example #3:

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

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

Example #4:

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

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

2341 12.5.2 Ref Examples

2342 **Example #1:**

2343 In this example the original query identifier is `xri://@a`. The first XRD contains an XRD-level
2344 Ref to `xri://@x*y`. The CanonicalID values are included to illustrate the CanonicalID
2345 verification rules in section 14.3.

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

2380 **Example #2:**

2381 In this example the original query identifier is `xri://@a*b*c`. The second XRD contains a SEP-
2382 level Ref in its authority resolution SEP to `xri://@x*y`. Note that because authority resolution is
2383 not complete when this Ref is encountered, it continues in the outer XRDS after the nested XRDS
2384 representing the Ref. *Note especially how the CanonicalIDs progress to satisfy the CanonicalID*
2385 *verification rules specified in section 14.3.*

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

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

Example #3:

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

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

```
2454            </XRD>
2455          <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2456            <Query>*b</Query>
2457            <ProviderID>xri://@!1</ProviderID>
2458            <CanonicalID>xri://@!1!2</CanonicalID>        ;XRDS #1 CID #2
2459            ...
2460            <Service>
2461              <Type>xri://$res*auth*($v*2.0)</Type>
2462              <URI>http://a.example.com/</URI>
2463            </Service>
2464          </XRD>
2465          <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2466            <Query>*c</Query>
2467            <ProviderID>xri://@!1!2</ProviderID>
2468            <CanonicalID>xri://@!1!2!3</CanonicalID>      ;XRDS #1 CID #3
2469            ...
2470            <Service>
2471              <Type>http://openid.net/signon/1.0</Type>
2472              <Ref>xri://@x*y</Ref>
2473            </Service>
2474          </XRD>
2475          <XRDS ref="xri://@x*y">
2476            <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2477              <Query>*x</Query>
2478              <ProviderID>xri://@</ProviderID>
2479              <CanonicalID>xri://@!7</CanonicalID>        ;XRDS #2 CID #1
2480              ...
2481              <Service>
2482                <Type>xri://$res*auth*($v*2.0)</Type>
2483                <URI>http://x.example.com/</URI>
2484              </Service>
2485            </XRD>
2486            <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
2487              <Query>*y</Query>
2488              <ProviderID>xri://@!7</ProviderID>
2489              <CanonicalID>xri://@!7!8</CanonicalID>      ;XRDS #2 CID #2
2490              ...
2491              <Service>
2492                <Type>xri://$res*auth*($v*2.0)</Type>
2493                <URI>http://y.example.com/</URI>
2494              </Service>
2495              <Service>
2496                <Type>http://openid.net/signon/1.0</Type>
2497                <URI>http://openid.example.com/</URI>
2498              </Service>
2499            </XRD>
2500          </XRDS>
2501        </XRDS>
2502
```

12.6 Recursion and Backtracking

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

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

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

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

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

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

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

13 Service Endpoint Selection

2540 The second phase of XRI resolution is called *service endpoint selection*. As noted in Figure 2, this
2541 phase is invoked automatically for each iteration of authority resolution after the first in order to
2542 select the Next Authority Resolution Service Endpoint as defined in section 9.1.9. It is also
2543 performed after authority resolution is complete if optional service endpoint selection is
2544 requested.

13.1 Processing Rules

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

Start SEP Selection

Input:
1) Set of all SEPs from final XRD
2) Path String
3) Service Type
4) Service Media Type

Branch to **SEP Selection Logic** Flowchart (Fig 9)

Output:
Set of selected SEPs

Selected set null? — Yes → Output error 241 SEP_NOT_FOUND

No

Select highest priority SEP

SEP Contains Redirect or Ref? — No → Output non-null set of selected SEPs

Yes

Input:
SEP

Branch to **Redirect & Ref Processing** flowchart (Fig 7)

Output:
Non-null set of selected SEPs or error

Error? — Yes → Output error

No → Output non-null set of selected SEPs

2547

2548 *Figure 8: Service endpoint (SEP) selection flowchart.*

2549 Following are the normative rules for the overall service endpoint selection process:

2550   1. The inputs for service endpoint selection are defined in Table 8.

2551   2. For the set of all service endpoints (`xrd:XRD/xrd:Service` elements) in the XRD,
2552   service endpoint selection MUST follow the logic defined in section 13.2. The output of
2553   this process MUST be either the null set or a selected set of one or more service
2554   endpoints.

2555   3. If, after applying the service endpoint selection logic, the selected set is null, this function
2556   MUST return the error `241 SEP_NOT_FOUND`.

2557   4. If, after applying the service endpoint selection logic, the selected set is not null and the
2558   highest priority selected service endpoint contains an
2559   `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref`
2560   element, it MUST first be processed as specified in section 12. This is a recursive call
2561   that will produce a nested XRDS document as defined in section 12.5.

2562

2563 13.2 Service Endpoint Selection Logic

2564 Selection of service endpoints (SEPs) within an XRD is managed using service endpoint
2565 selection elements (SELs). As shown in Figure 9 (non-normative), the selection process first
2566 applies SEL matching rules (section 13.3), followed by SEP matching rules (section 13.4), to the
2567 set of all SEPs in the XRD. It then applies SEP selection rules (section 13.5) to determine the
2568 final output.

2569
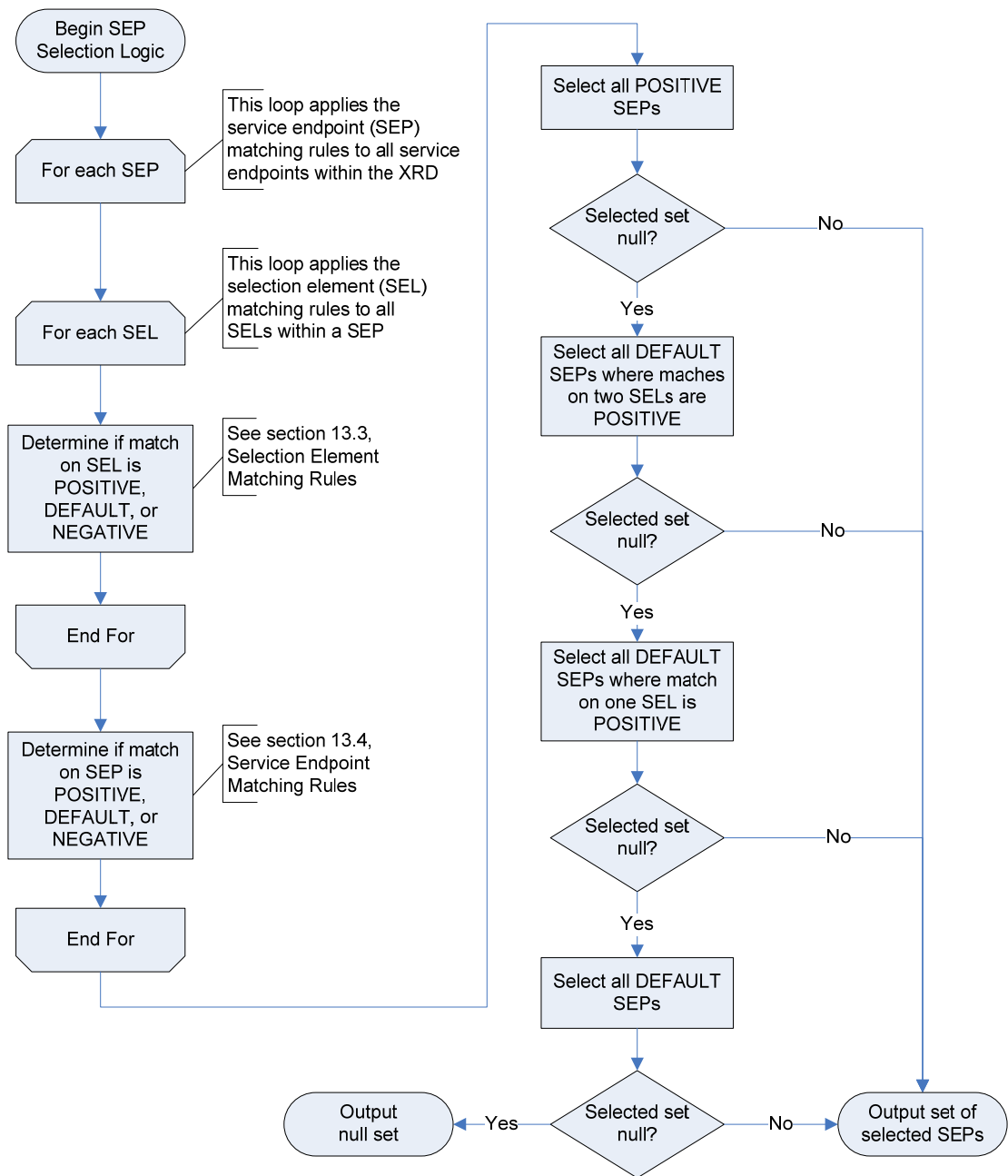
2570 *Figure 9: Service endpoint (SEP) selection logic flowchart.*

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

2572 13.3 Selection Element Matching Rules

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

2574 13.3.1 Selection Element Match Options

2575 As defined in section 4.2.6, there are three categories of service endpoint selection elements:
2576 `xrd:Type`, `xrd:Path`, and `xrd:MediaType`. Within each service endpoint, there is a match
2577 option for each of the three categories of selection elements. Matches are tri-state: the three
2578 options and their corresponding precedence order are defined in Table 24:

Match Option	Match Condition	Precedence
POSITIVE	A successful match based on the value of the `match` attribute as defined in 13.3.2 OR a successful match based the contents of the selection element as defined in sections 13.3.6 - 13.3.8.	1
DEFAULT	The value of the `match` attribute is `default` OR there is no instance of this type of selection element contained in the service endpoint as defined in section 13.3.3.	0
NEGATIVE	The selection element does not satisfy either condition above.	-1

2579 *Table 24: Match options for selection elements.*

2580 The Precedence order is used in the Multiple Selection Element Matching Rule (section 13.3.5).

2581 IMPORTANT: Failure of a POSITIVE match does not necessarily mean a NEGATIVE match; it
2582 may still qualify as a DEFAULT match.

2583 13.3.2 The Match Attribute

2584 All three service endpoint selection elements accept the optional `match` attribute. This attribute
2585 gives XRDS authors precise control over selection of SEPs based on the QXRI and other service
2586 endpoint selection parameters. An enumerated list of the values for the `match` attribute is defined
2587 in Table 25. If the `match` attribute is present with one of these values, the contents of the
2588 selection element MUST be ignored, and the corresponding matching rule MUST be applied. If
2589 the `match` attribute is absent or has any other value, the rules in this section do not apply.

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

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

13.3.3 Absent Selection Element Matching Rule

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

13.3.4 Empty Selection Element Matching Rule

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

13.3.5 Multiple Selection Element Matching Rule

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

13.3.6 Type Element Matching Rules

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

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

2. Prior to comparsion (and only for the purpose of comparison), the values of the Service Type parameter and the `xrd:XRD/xrd:Service/xrd:Type` element SHOULD be normalized according to the requirements of their identifier scheme. In particular, if an XRI, IRI, or URI uses hierarchical syntax and does not include a local part (a path and/or query component) after the authority component, a trailing forward slash after the authority component MUST NOT be considered significant in comparisions. In all other cases, a trailing forward slash MUST be considered significant in comparisons unless this rule is overridden by scheme-specific comparision rules.

3. To result in a POSITIVE match on this selection element, the values MUST be equivalent according to the equivalence rules of the applicable identifier scheme. Any other result is a NEGATIVE match on this selection element.

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

13.3.7 Path Element Matching Rules

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

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

2. Prior to comparison, the leading forward slash separating an XRI authority component from the path component MUST be prepended to the Path String. Any subsequent forward slash, including trailing forward slashes, MUST be significant in comparisions.

3. The contents of the `xrd:XRD/xrd:Service/xrd:Path` element SHOULD include the leading forward slash separating the XRI authority component from the path. If it does not, one MUST be prepended prior to comparision.

4. Equivalence comparison SHOULD be performed using Caseless Matching as defined in section 3.13 of **[Unicode]**.

5. To result in a POSITIVE match on this selection element, the value of the Path String MUST be a *subsegment stem match* with the contents of the `xrd:XRD/xrd:Service/xrd:Path` element. A subsegment stem match is defined as the entire Path String being character-for-character equivalent with any continuous sequence of subsegments or segments (including empty subsegments and empty segments) in the contents of the Path element beginning from the most significant (leftmost) subsegment. Subsegments and segments are formally defined in **[XRISyntax]**. Any other result MUST be a NEGATIVE match on this selection element.

2652   Examples of this rule are shown in Table 26.

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

2653   *Table 26: Examples of applying the Path element matching rules.*

2654

13.3.8 MediaType Element Matching Rules

2655

2656 The following rules apply to matching the value of the input Service Media Type parameter with
2657 the contents of of a non-empty `xrd:XRD/xrd:Service/xrd:MediaType` element when its
2658 `match` attribute is absent.

2659　　1.　The values of the Service Media Type parameter and the `xrd:MediaType` element
2660　　　　SHOULD be normalized according to the rules for media types in section 3.7 of
2661　　　　**[RFC2616]** prior to input. (The rules are that media type and media type parameter
2662　　　　names are case-insensitive, but parameter values may or may not be case-sensitive
2663　　　　depending on the semantics of the parameter name. XRI Resolution Output Format
2664　　　　parameters and subparameters are all case-insensitive.) XRI resolvers MAY perform
2665　　　　normalization of these values but MUST NOT be required to do so.

2666　　2.　To be a POSITIVE match on this selection element, the values MUST be character-for-
2667　　　　character equivalent. Any other result is a NEGATIVE match on this selection element.

13.4 Service Endpoint Matching Rules

2668

2669 The next set of matching rules govern the matching of service endpoints based on the matches of
2670 the selection elements they contain.

13.4.1 Service Endpoint Match Options

2671

2672 For each service endpoint in an XRD, there are three match options as defined in Table 27:

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

2673　*Table 27: Match options for service endpoints.*

13.4.2 Select Attribute Match Rule

2674

2675 All three service endpoint selection elements accept the optional `select` attribute. This attribute
2676 is a Boolean value used to govern matching of the containing service endpoint according to the
2677 following rule. If service endpoint contains a selection element with a POSITIVE match as defined
2678 in section 13.3, and the value of this selection element's `select` attribute is TRUE, the service
2679 endpoint automatically MUST be a POSITIVE match, i.e., all other selection elements for this
2680 service endpoint MUST be ignored.

13.4.3 All Positive Match Rule

2681

2682 If a service endpoint has a POSITIVE match on all three categories of selection elements
2683 (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) as defined in section 13.3, the service endpoint
2684 MUST be a POSITIVE match. If even one of the three selection element match types is not
2685 POSITIVE, this rule fails.

13.4.4 Default Match Rule

2686

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

13.5 Service Endpoint Selection Rules

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

13.5.1 Positive Match Rule

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

13.5.2 Default Match Rule

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

1. The service endpoints in the DEFAULT set that have two POSITIVE selection element matches MUST be selected.

2. If the previous set is empty, the service endpoints in the DEFAULT set that have one POSITIVE selection element match MUST be selected.

3. If the previous set is empty, all service endpoints in the DEFAULT set MUST be selected.

4. If the previous set is empty, no service endpoint is selected and the return set is null.

13.6 Pseudocode

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

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

- Postive.x (where x = Type, Path, or MediaType)

- Default.x (where x = Type, Path, or MediaType)

- Present.x (where x = Type, Path, or MediaType)

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

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

1. Set the SEL match flags according to the rules specified in section 13.3;

2. Process the SEL match flags to apply the SEP matching rules specified in section 13.4;

3. Apply the positive SEP selection rule specified in section 13.5.1.

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

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

```
2729

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

2792 13.7 Construction of Service Endpoint URIs

2793 The final step in the service endpoint selection process is construction of the service endpoint
2794 URI(s). This step is necessary if either:

2795 • The resolution output format is a URI List.

2796 • Automatic URI construction is requested using the `uric` parameter.

2797 13.7.1 The append Attribute

2798 The `append` attribute of a `xrd:XRD/xrd:Service/xrd:URI` element is used to specify how
2799 the final URI is constructed. The values of this attribute are shown in Table 28.

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

2800 *Table 28: Values of the `append` attribute and the corresponding QXRI component to append.*

2801 If the `append` attribute is absent, the default value is `none`. Following are the rules for
2802 construction of the final service endpoint URI based on the value of the `append` attribute.

2803 IMPORTANT: Implementers must follow these rules exactly in order to give XRDS authors
2804 precise control over construction of service endpoint URIs.

2805 1. If the value is `none`, the exact contents of the `xrd:URI` element MUST be returned
2806   directly without any further processing.

2807 2. For any other value, the exact value in URI-normal form of the QXRI component specified
2808   in Table 28, *including any leading delimiter(s)* and *without any additional escaping or
2809   percent encoding* MUST be appended directly to the exact contents of the `xrd:URI`
2810   element *including any trailing delimiter(s)*. If the value of the QXRI component specified in
2811   Table 28 consists of only a leading delimiter, then this value MUST be appended
2812   according to these rules. If the value of the QXRI component specified in Table 28 is null,
2813   then the contents of the `xrd:URI` element MUST be returned directly exactly as if the
2814   value of the `append` attribute was `none`.

2815   3.   If any HXRI query parameters for proxy resolution were added to an existing QXRI query
2816   component as defined in section 11.3, these query parameters MUST be removed prior
2817   to performing the append operation as also defined in section 11.3. In particular, if after
2818   removal of these query parameters the QXRI query component consists of only *a string
2819   of one or more question marks* (the delimiting question mark plus zero or more additional
2820   question marks) then *exactly one question mark* MUST also be removed. This preserves
2821   the query component of the original QXRI if it was null or contained only question marks.

2822   IMPORTANT: Construction of HTTP(S) URIs for authority resolution service endpoints is defined
2823   in section 9.1.10. Note that this involves an additional step taken after all URI construction steps
2824   specified in this section are complete. In other words, if the URI element of an authority resolution
2825   service endpoint includes an `append` attribute, the Next Authority Resolution Service URI MUST
2826   be fully constructed according to the algorithm in this section before appending the Next Authority
2827   String as defined in section 9.1.10.

2828   WARNING: Use of any value of the `append` attribute other than `authority` on the URI element
2829   for an authority resolution service endpoint is NOT RECOMMENDED due to the complexity it
2830   introduces.

2831 13.7.2 The uric Parameter

2832   The `uric` subparameter of the Resolution Output Format is used to govern whether a resolver
2833   should perform construction of the URI automatically on behalf of a consuming application.
2834   Following are the processing rules for this parameter:

2835   1.   If `uric=true`, a resolver MUST apply the URI construction rules specified in section
2836   13.7.1 to each `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD in the
2837   resolution chain. Note that this step is identical to the processing a resolver must perform
2838   to output a URI list.

2839   2.   The resolver MUST replace the value of each `xrd:XRD/xrd:Service/xrd:URI`
2840   element in the final XRD with the fully constructed URI value.

2841   3.   The resolver MUST subsequently remove the `append` attribute from each
2842   `xrd:XRD/xrd:Service/xrd:URI` element in the final XRD.

2843   4.   If `uric=false` or the parameter is absent or empty, a resolver MUST NOT perform any
2844   of the processing specified in this section.

14 Synonym Verification

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

14.1 Redirect Verification

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

1. If resolution of the Redirect succeeds, the resolver MUST first verify that the set of XRD synonym elements (as specified in section 5.2) contained in the new XRD are *equivalent to or a subset of* those contained in the XRD containing the Redirect.

2. Secondly, the resolver MUST verify that the content of each synonym element contained in the new XRD is exactly equivalent to the content of the corresponding element in the XRD containing the Redirect.

3. If either rule above fails, the resolver MUST stop and return the error `253 REDIRECT_VERIFY_FAILED` in the XRD where the error occurred or as a plain text error message as defined in section 15.

For examples see section 12.5.1.

14.2 EquivID Verification

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

1. First request resolution for the original query identifier with CanonicalID verification enabled (`cid=true`).

2. From the final XRD in the resolution chain, select the EquivID for which verification is desired.

3. Request resolution of the EquivID identifier.

4. From the final XRD in this second resolution chain, determine if there is either: a) a `xrd:XRD/xrd:EquivID` element, or b) a `xrd:XRD/xrd:CanonicalEquivID` element whose value matches the verified CanonicalID of the original query identifier. If there is a match, the EquivID is verified; otherwise it is not verified.

Example:

- Fully-Qualified Query Identifier: `http://example.com/user`

- Asserted EquivID: `xri://=!1000.c78d.402a.8824.bf20`

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

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

2888 Second XRDS (for `xri://=!1000.c78d.402a.8824.bf20`):

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

2901 The EquivID is verified because the XRD in the second XRDS asserts an EquivID backpointer to
2902 the CanonicalID of the XRD in the first XRDS.

14.3 CanonicalID Verification

2904 XRI resolvers automatically perform verification of CanonicalID and CanonicalEquivID synonyms
2905 unless this function is explicitly turned off using the Resolution Output Format subparameter `cid`.
2906 The following synonym verification MUST be applied by an XRI resolver if `cid=true` or the
2907 parameter is absent or empty, and MUST NOT be applied if `cid=false`.

2908   1. If the value of the `xrd:XRD/xrd:CanonicalID` element is an HTTP(S) URI, it MUST
2909   be verified as specified in section 14.3.1.

2910   2. If the value of the `xrd:XRD/xrd:CanonicalID` element is an XRI, it MUST be verified
2911   as specified in section 14.3.2.

2912   3. If the value of the `xrd:XRD/xrd:CanonicalID` element is any other identifier,
2913   CanonicalID verification fails and the resolver MUST return the CanonicalID verification
2914   status specified in section 14.3.4.

2915   4. If CanonicalID verification succeeds but the final XRD in the resolution chain also
2916   contains a `xrd:XRD/xrd:CanonicalEquivID` element, it MUST also be verified as
2917   specified in section 14.3.3, and the resolver MUST return the CanonicalEquivID
2918   verification status as specified in section 14.3.4.

2919   5. In all cases, since synonym verification depends on trusting each authority in the
2920   resolution chain, trusted resolution (section 10) SHOULD be used with either
2921   `https=true` or `saml=true` or both to provide additional assurance of the authenticity of
2922   the results.

2923 IMPORTANT: There is no guarantee that all XRDs that describe the same target resource will
2924 return the same verified CanonicalID or CanonicalEquivID. Different parent authorities may assert
2925 different CanonicalIDs or CanonicalEquivIDs for the same target resource and all of these may all
2926 be verifiable. In addition, due to Redirect and Ref processing, the verified CanonicalID or
2927 CanonicalEquivID returned for an XRI MAY differ depending on the resolution input parameters.
2928 For example, as described in section 12, a request for a specific service endpoint type may
2929 trigger processing of a Redirect or Ref resulting in a nested XRDS document. The final XRD in
2930 the nested XRDS document may come from a different parent authority and have a different but
2931 still verifiable CanonicalID or CanonicalEquivID.

2932

14.3.1 HTTP(S) URI Verification Rules

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

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

2. The query identifier MUST be resolved as specified in section 6.

3. The asserted CanonicalID synonym MUST be an HTTP(S) URI equivalent to: a) the fully-qualified query identifier, or b) the fully-qualified query identifier plus a valid fragment as defined by **[RFC3986]**.

See the example in section 14.3.5.

14.3.2 XRI Verification Rules

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

1. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID` element MUST consist of two parts:

   1) The value of the `xrd:XRD/xrd:CanonicalID` element for the community root authority as configured in the XRI resolver or asserted in a self-describing XRD from the community root authority (or via another equivalent mechanism as described in section 9.1.6).

   2) One additional XRI subsegment as defined in **[XRISyntax]**. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element for the community root authority was `@`, then the following would all be verified values for the `xrd:XRD/xrd:CanonicalID` element in the first XRD in the resolution chain: `@!1`, `@!1234`, `@!example`, `@example` (note that `@example` is not recommended because it is not a persistent identifier).

2. For each subsequent XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID` element MUST consist of the value the `xrd:XRD/xrd:CanonicalID` element of the preceding XRD in the same XRDS document plus one additional XRI subsegment. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element asserted in an XRD is `@!1!2!3`, then the value of the `xrd:XRD/xrd:CanonicalID` element in the immediately preceding XRD in the same XRDS document must be `@!1!2`.

3. If Redirect or Ref processing is required during resolution as specified in section 12, the rules above MUST also apply for each nested XRDS document.

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

14.3.3 CanonicalEquivID Verification

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

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

2977 To verify that a CanonicalEquivID is an authorized synonym for a verified CanonicalEquivID, a
2978 resolver MUST verify that either: a) the value of the CanonicalEquivID element is character-by-
2979 character equivalent to the verified CanonicalID (since both appear in the same XRD, all other
2980 normalization rules are waived), or b) that all the following tests are successful:

    1. The asserted CanonicalEquivID value MUST be a valid HTTP(S) URI or XRI.

2982   2. The asserted CanonicalEquivID value MUST resolve successfully to an XRDS document
2983   according to the rules in this specification *using the same resolution parameters as in the*
2984   *original resolution request*.

2985   3. The CanonicalID in the final XRD of the resolved XRDS document MUST be verified and
2986   MUST be equivalent to the asserted CanonicalEquivID.

2987   4. The final XRD in the resolved XRDS document MUST contain either an EquivID or a
2988   CanonicalEquivID "backpointer" whose value is equivalent to the verified CanonicalID in
2989   the XRD asserting the CanonicalEquivID.

2990 SPECIAL SECURITY CONSIDERATION: See section 5.2.2 regarding the rules for provisioning
2991 of `xrd:XRD/xrd:EquivID` and `xrd:XRD/xrd:CanonicalEquivID` elements in an XRD.

14.3.4 Verification Status Attributes

2993 If CanonicalID verification is performed, an XRI resolver MUST return the CanonicalID and
2994 CanonicalEquivID verification status using an attribute of the `xrd:XRD/xrd:Status` element in
2995 each XRD in the output as follows:

2996   1. CanonicalID verification MUST be reported using the `cid` attribute.

2997   2. CanonicalEquivID verification MUST be reported using the `ceid` attribute.

2998   3. Both attributes accept four enumerated values: `absent` if the element is not present, `off`
2999   if verification is not performed, `verified` if the element is verified, and `failed` if
3000   verification fails.

3001   4. The `off` value applies to both elements if CanonicalID verification is not performed
3002   (`cid=false`).

3003   5. The `off` value applies to the CanonicalEquivID element in any XRD before the final XRD
3004   if CanonicalID verification is performed (`cid=true`), because a resolver only verifies this
3005   element in the final XRD.

3006   6. If `cid=true` and verification of any CanonicalID element fails, *verification of all*
3007   *CanonicalIDs in all subsequent XRDs in the same XRDS document MUST fail*.

3008 From these verification status attributes, a consuming application can confirm on every XRD in
3009 the XRDS document whether the CanonicalID is present and has been verified. In addition, for
3010 the final XRD in the XRDS document, it can confirm whether the CanonicalEquivID element is
3011 present and has been verified.

3012

14.3.5 Examples

Example #1:

- Fully-Qualified Query Identifier: `http://example.com/user`

- Asserted CanonicalID: `http://example.com/user#1234`

XRDS (simplified for illustration purposes):

```
<XRDS ref="http://example.com/user">
  <XRD>
    <CanonicalID>http://example.com/user#1234</CanonicalID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

Example #2:

- Fully-Qualified Query Identifier: `=example.name*delegate.name`

- Asserted CanonicalID: `=!1000.62b1.44fd.2855!1234`

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

```
<XRDS ref="xri://=example.name*delegate.name">
  <XRD>
    <Query>*example.name</Query>
    <ProviderID>xri://=</ProviderID>
    <LocalID>!1000.62b1.44fd.2855</LocalID>
    <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
    <Service>
       <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
       <Type>xri://$res*auth*($v*2.0)</Type>
       <MediaType>application/xrds+xml</MediaType>
       <URI priority="10">http://resolve.example.com</URI>
       <URI priority="15">http://resolve2.example.com</URI>
       <URI>https://resolve.example.com</URI>
    </Service>
    ...
  </XRD>
  <XRD>
    <Query>*delegate.name</Query>
    <ProviderID>xri://=!1000.62b1.44fd.2855</ProviderID>
    <LocalID>!1234</LocalID>
    <CanonicalID>xri://=!1000.62b1.44fd.2855!1234</CanonicalID>
    <Service priority="1">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

3062 **Example #3:**

3063 • Fully-Qualified Query Identifier: `http://example.com/user`

3064 • Asserted CanonicalID: `http://example.com/user`

3065 • Asserted CanonicalEquivID: `https://different.example.net/path/user`

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

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

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

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

3090 The CanonicalEquivID asserted in the first XRDS satisifies the verification rules in section 14.3.3
3091 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of
3092 the first XRDS.

3093

3094 **Example #4:**

3095 • Fully-Qualified Query Identifier: `http://example.com/user`

3096 • Asserted CanonicalID: `http://example.com/user`

3097 • Asserted CanonicalEquivID: `=!1000.62b1.44fd.2855`

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

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

3109

3110 XRDS (for `xri://=!1000.62b1.44fd.2855`):

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

3123 The CanonicalEquivID asserted in the first XRDS satisifies the verification rules in section 14.3.3
3124 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of
3125 the first XRDS.

3126

3127 **Example #5:**

3128 • Fully-Qualified Query Identifier: `=example.name`

3129 • Asserted CanonicalID: `xri://=!1000.62b1.44fd.2855`

3130 • Asserted CanonicalEquivID: `https://example.com/user`

3131 First XRDS (for `=example.name`):

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

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

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

3156 The CanonicalEquivID asserted in the first XRDS satisifies the verification rules in section 14.3.3
3157 because it resolves to a second XRDS that asserts an EquivID backpointer to the CanonicalID of
3158 the first XRDS.

3159

3160

3161 **Example #6:**

3162 • Fully-Qualified Query Identifier: =example.name*delegate.name

3163 • Asserted CanonicalID: `xri://=!1000.62b1.44fd.2855!1234`

3164 • Asserted CanonicalEquivID: `@!1000.f3da.9056.aca3!5555`

3165 First XRDS (for =example.name*delegate.name):

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

3196 • Second XRDS (for @!1000.f3da.9056.aca3!5555):

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

```
3217          <CanonicalID>xri://@!1000.f3da.9056.aca3!5555</CanonicalID>
3218          <Service priority="1">
3219             ...
3220          </Service>
3221          ...
3222        </XRD>
3223      </XRDS>
```

3224   The CanonicalEquivID asserted in the final XRD of the first XRDS satisifies the verification rules
3225   in section 14.3.3 because it resolves to a second XRDS whose final XRD asserts an EquivID
3226   backpointer to the CanonicalID of the final XRD in the first XRDS.

15 Status Codes and Error Processing

15.1 Status Elements

XRDS architecture uses two XRD elements for status reporting:

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

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

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

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

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

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

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

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

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

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

15.2 Status Codes

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

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

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

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

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

3266 • x0x: General error that may take place during any phase of resolution.

3267 • x1x: Input error

3268 • x2x: Generic authority resolution error.

3269 • x3x: Trusted authority resolution error.

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

3271 • x5x: Redirect error.

3272 • x6x: Ref error.

3273 The full list of XRI resolution status codes is defined in Table 29.

3274

Code	Symbolic Status	Phase(s)	Description
100	SUCCESS	Any	Operation was successful.
200	PERM_FAIL	Any	Generic permanent failure.
201	NOT_IMPLEMENTED	Any	The requested function (trusted resolution, service endpoint selection) is not implement by the resolver.
202	LIMIT_EXCEEDED	Any	A locally configured resource limit was exceeded. Examples: number of Redirect or Refs to follow, number of XRD elements that can be handled, size of an XRDS document.
210	INVALID_INPUT	Input	Generic input error.
211	INVALID_QXRI	Input	Input QXRI does not conform to XRI syntax.
212	INVALID_OUTPUT_FORMAT	Input	Input Resolution Output Format is invalid.
213	INVALID_SEP_TYPE	Input	Input Service Type is invalid.
214	INVALID_SEP_MEDIA_TYPE	Input	Input Service Media Type is invalid.
215	UNKNOWN_ROOT	Input	Community root specified in QXRI is not configured in the resolver.
220	AUTH_RES_ERROR	Authority resolution	Generic authority resolution error.
221	AUTH_RES_NOT_FOUND	Authority resolution	The subsegment cannot be resolved due to a missing authority resolution service endpoint in an XRD.
222	QUERY_NOT_FOUND	Authority resolution	Responding authority does not have an XRI matching the query.
223	UNEXPECTED_XRD	Authority resolution	Value of the xrd:Query element does not match the subsegment requested.
224	INACTIVE	Authority resolution	The query XRI has been assigned but the authority does not provide resolution metadata.

3275

3276

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

3277 *Table 29: Error codes for XRI resolution.*

15.3 Status Context Strings

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

15.4 Returning Errors in Plain Text or HTML

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

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

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

15.5 Error Handling in Recursing and Proxy Resolution

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

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

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

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

16 Use of HTTP(S)

16.1 HTTP Errors

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

16.2 HTTP Headers

16.2.1 Caching

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

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

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

16.2.2 Location

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

16.2.3 Content-Type

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

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

16.3 Other HTTP Features

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

16.4 Caching and Efficiency

16.4.1 Resolver Caching

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

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

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

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

16.4.2 Synonyms

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

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

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

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

17 Extensibility and Versioning

17.1 Extensibility

17.1.1 Extensibility of XRDs

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

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

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

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

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

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

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

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

3436 `xrd:XRD/xrd:Type` element. However XRD processors MAY ignore instances of this element
3437 and process the XRD as specified in this document.

17.1.2 Other Points of Extensibility

3439 The use of HTTP(S), XML, XRIs, and URIs in the design of XRDS documents, XRD elements,
3440 and XRI resolution architecture provides additional specific points of extensibility:

3441 • Specification of new resolution service types or other service types using XRIs, IRIs, or URIs
3442 as values of the `xrd:Type` element.

3443 • Specification of new resolution output formats or features using media types and media type
3444 parameters as values of the `xrd:MediaType` element as defined in **[RFC2045]** and
3445 **[RFC2046]**.

3446 • HTTP negotiation of content types, language, encoding, etc. as defined by **[RFC2616]**.

3447 • Use of HTTP redirects (3XX) or other response codes defined by **[RFC2616]**.

3448 • Use of cross-references within XRIs, particularly for associating new types of metadata with a
3449 resource. See **[XRISyntax]** and **[XRIMetadata]**.

17.2 Versioning

3451 Versioning of the XRI specification set is expected to occur infrequently. Should it be necessary,
3452 this section describes versioning guidelines.

3453 In general, this specification follows the same versioning guidelines as established in section
3454 4.2.1 of **[SAML]**:

3455 *In general, maintaining namespace stability while adding or changing the content of a*
3456 *schema are competing goals. While certain design strategies can facilitate such changes,*
3457 *it is complex to predict how older implementations will react to any given change, making*
3458 *forward compatibility difficult to achieve. Nevertheless, the right to make such changes in*
3459 *minor revisions is reserved, in the interest of namespace stability. Except in special*
3460 *circumstances (for example, to correct major deficiencies or to fix errors),*
3461 *implementations should expect forward-compatible schema changes in minor revisions,*
3462 *allowing new messages to validate against older schemas.*

3463 *Implementations SHOULD expect and be prepared to deal with new extensions and*
3464 *message types in accordance with the processing rules laid out for those types. Minor*
3465 *revisions MAY introduce new types that leverage the extension facilities described in [this*
3466 *section]. Older implementations SHOULD reject such extensions gracefully when they*
3467 *are encountered in contexts that dictate mandatory semantics.*

17.2.1 Version Numbering

3469 Specifications from the OASIS XRI Technical Committee use a Major and Minor version number
3470 expressed in the form Major.Minor. The version number MajorB.MinorB is higher than the version
3471 number $Major_A.Minor_A$ if and only if:

3472 $Major_B > Major_A$ OR (($Major_B = Major_A$) AND $Minor_B > Minor_A$)

17.2.2 Versioning of the XRI Resolution Specification

3474 New releases of the XRI Resolution specification may specify changes to the resolution protocols
3475 and/or the XRD schema in Appendix B. When changes affect either of these, the resolution
3476 service type version number will be changed. Where changes are purely editorial, the version
3477 number will not be changed.

3478   In general, if a change is backward-compatible, the new version will be identified using the
3479   current major version number and a new minor version number. If the change is not backward-
3480   compatible, the new version will be identified with a new major version number.

3481 17.2.3 Versioning of Protocols

3482   The protocols defined in this document may also be versioned by future releases of the XRI
3483   Resolution specification. If these protocols are not backward-compatible with older
3484   implementations, they will be assigned a new XRI with a new version identifier for use in
3485   identifying their service type in XRDs. See section 3.1.2.

3486   Note that it is possible for version negotiation to happen in the protocol itself. For example, HTTP
3487   provides a mechanism to negotiate the version of the HTTP protocol being used. If and when an
3488   XRI resolution protocol provides its own version-negotiation mechanism, the specification is likely
3489   to continue to use the same XRI to identify the protocol as was used in previous versions of the
3490   XRI Resolution specification.

3491 17.2.4 Versioning of XRDs

3492   The `xrd:XRDS` document element is intended to be a completely generic container, i.e., to have
3493   no specific knowledge of the elements it may contain. Therefore it has no version indicator, and
3494   can remain stable indefinitely because there is no need to version its namespace.

3495   The `xrd:XRD` element has a `version` attribute. This attribute is OPTIONAL for this version of
3496   the XRI resolution specification (version 2.0). This attribute will be REQUIRED for all future
3497   versions of this specification. When used, the value of this attribute MUST be the exact numeric
3498   version value of the XRI Resolution specification to which its containing elements conform.

3499   When new versions of the XRI Resolution specification are released, the namespace for the XRD
3500   schema may or may not be changed. If there is a major version number change, the namespace
3501   for the `xrd:XRD` schema is likely to change. If there is only a minor version number change, the
3502   namespace for the `xrd:XRD` schema may remain unchanged.

3503   Note that conformance to a specific XRD version does not preclude an author from including
3504   extension elements from a different namespace in the XRD. See section 17.1 above.

18 Security and Data Protection

3505

3506 Significant portions of this specification deal directly with security issues; these will not be
3507 summarized again here. In addition, basic security practices and typical risks in resolution
3508 protocols are well-documented in many other specifications. Only security considerations directly
3509 relevant to XRI resolution are included here.

18.1 DNS Spoofing or Poisoning

3510

3511 When XRI resolution is deployed to use HTTP URIs or other URIs which include DNS names, the
3512 accuracy of the XRI resolution response may be dependent on the accuracy of DNS queries. For
3513 those deployments where DNS is not trusted, the resolution infrastructure may be deployed with
3514 HTTP URIs that use IP addresses in the authority portion of HTTP URIs and/or with the trusted
3515 resolution mechanisms defined by this specification. Resolution results obtained using trusted
3516 resolution can be evaluated independently of DNS resolution results. While this does not solve
3517 the problem of DNS spoofing, it does allow the client to detect an error condition and reject the
3518 resolution result as untrustworthy. In addition, **[DNSSEC]** may be considered if DNS names are
3519 used in HTTP URIs.

18.2 HTTP Security

3520

3521 Many of the security considerations set forth in HTTP/1.1 **[RFC2616]** apply to XRI Resolution
3522 protocols defined here. In particular, confidentiality of the communication channel is not
3523 guaranteed by HTTP. Server-authenticated HTTPS should be used in cases where confidentiality
3524 of resolution requests and responses is desired.

3525 Special consideration should be given to proxy and caching behaviors to ensure accurate and
3526 reliable responses from resolution requests. For various reasons, network topologies increasingly
3527 have transparent proxies, some of which may insert VIA and other headers as a consequence, or
3528 may even cache content without regard to caching policies set by a resource's HTTP authority.

3529 Implementations of XRI Proxies and caching authorities should also take special note of the
3530 security recommendations in HTTP/1.1 **[RFC2616]** section 15.7.

18.3 SAML Considerations

3531

3532 SAML trusted authority resolution must adhere to the rules defined by the SAML 2.0 Core
3533 Specification **[SAML]**. Particularly noteworthy are the XML Transform restrictions on XML
3534 Signature and the enforcement of the SAML Conditions element regarding the validity period.

18.4 Limitations of Trusted Resolution

3535

3536 While the trusted resolution protocols specified in this document provide a way to verify the
3537 integrity of a successful XRI resolution, it may not provide a way to verify the integrity of a
3538 resolution failure. Reasons for this limitation include the prevalence of non-malicious network
3539 failures, the existence of denial-of-service attacks, and the ability of a man-in-the-middle attacker
3540 to modify HTTP responses when resolution is not performed over HTTPS.

3541 Additionally, there is no revocation mechanism for the keys used in trusted resolution. Therefore,
3542 a signed resolution's validity period should be limited appropriately to mitigate the risk of an
3543 incorrect or invalid resolution.

18.5 Synonym Verification

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

18.6 Redirect and Ref Management

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

18.7 Community Root Authorities

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

18.8 Caching Authorities

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

18.9 Recursing and Proxy Resolution

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

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

18.10 Denial-Of-Service Attacks

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

A. Acknowledgments

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

- William Barnhill, Booz Allen and Hamilton
- Dave McAlpin, Epok
- Chetan Sabnis, Epok
- Peter Davis, Neustar
- Victor Grey, PlaNetwork
- Mike Lindelsee, Visa International
- Markus Sabadello, XDI.org
- John Bradley
- Kermit Snelson

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

- Geoffrey Strongin, Advanced Micro Devices
- Ajay Madhok, AmSoft Systems
- Dr. XiaoDong Lee, China Internet Network Information
- Nat Sakimura, Nomura Research
- Owen Davis, PlaNetwork
- Fen Labalme, PlaNetwork
- Marty Schleiff, The Boeing Company
- Dave Wentker, Visa International
- Paul Trevithick

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

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

3611 B. RelaxNG Schema for XRDS and XRD

3612 Following are the locations of the normative RelaxNG compact schema files for XRDS and XRD
3613 as defined by this specification:

3614 • **xrds.rnc**: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrds.mc

3615 • **xrd.rnc**: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrd.mc

3616 IMPORTANT: The **xrd.rnc** schema does NOT include deprecated attribute values that are
3617 recommended for backwards compatibility. See the highlighted Backwards Compatibility notes in
3618 sections 9.1.1 and 13.3.2 for more details.

3619 Listings of these files are provided in this appendix for reference but are non-normative.

3620 **xrds.rnc**

```
3621  namespace xrds = "xri://$xrds"
3622  namespace xrd = "xri://$xrd*($v*2.0)"
3623  namespace local = ""
3624  datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
3625
3626  any.element =
3627    element * {
3628      ( attribute* { text }*
3629        | text
3630        | any.element )*
3631    }
3632
3633  any.external.element =
3634    element * - ( xrd:XRD | xrds:XRDS )  {
3635      ( attribute * { text } *
3636        | text
3637        | any.element )*
3638    }
3639
3640  other.attribute = attribute * - ( local:* ) { text }
3641
3642  start = XRDS
3643
3644  XRDS = element xrds:XRDS {
3645      other.attribute *,
3646      ( attribute ref { xs:anyURI } | attribute redirect { xs:anyURI } )?,
3647      ( any.external.element  | XRDS | external "xrd.rnc" )*
3648  }
3649
```

3650 **xrd.rnc**

```
3651  default namespace = "xri://$xrd*($v*2.0)"
3652  namespace xrd = "xri://$xrd*($v*2.0)"
3653  namespace saml = "urn:oasis:names:tc:SAML:2.0:assertion"
3654  namespace ds = "http://www.w3.org/2000/09/xmldsig#"
3655  namespace local = ""
3656
3657  datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"
3658
3659  start = XRD
3660
3661  anyelementbody =
3662      (attribute * {text}
3663        | text
3664        | element * {anyelementbody} )*
3665
3666
```

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

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

3806 C. XML Schema for XRDS and XRD

3807 Following are the locations of the non-normative W3C XML Schema files for XRDS and XRD as
3808 defined by this specification. Note that these are provided for reference only as they are not able
3809 to fully express the extensibility semantics of the RelaxNG versions.

3810 • **xrds.xsd**: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrds.xsd

3811 • **xrd.xsd**: http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cd03/xrd.xsd

3812 IMPORTANT: The **xrd.xsd** schema does NOT include deprecated attribute values that are
3813 recommended for backwards compatibility. See the highlighted Backwards Compatibility notes in
3814 sections 9.1.1 and 13.3.2 for more details.

3815 Listings of these files are provided in this appendix for reference.

3816 **xrds.xsd**

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

3847 **xrd.xsd**

```
3848  <?xml version="1.0" encoding="UTF-8"?>
3849  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3850  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xrd="xri://$xrd*($v*2.0)"
3851  targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified">
3852      <!-- Utility patterns -->
3853      <xs:attributeGroup name="otherattribute">
3854          <xs:anyAttribute namespace="##other" processContents="lax"/>
3855      </xs:attributeGroup>
3856      <xs:group name="otherelement">
3857          <xs:choice>
3858              <xs:any namespace="##other" processContents="lax"/>
3859              <xs:any namespace="##local" processContents="lax"/>
3860          </xs:choice>
3861      </xs:group>
3862
```

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

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

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

D. Media Type Definition for application/xrds+xml

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

Type name: application

Subtype name: xrds+xml

Required parameters: None

Optional parameters: See Table 6 of this document.

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

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

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: Applications conforming to this specification use this media type.

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

Intended usage: COMMON

Restrictions on usage: None

Author: OASIS XRI TC

Change controller: OASIS XRI TC

E. Media Type Definition for application/xrd+xml

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

Type name: application

Subtype name: xrd+xml

Required parameters: None

Optional parameters: See Table 6 of this document.

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

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

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification.

Applications that use this media type: Applications conforming to this specification use this media type.

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

Intended usage: COMMON

Restrictions on usage: None

Author: OASIS XRI TC

Change controller: OASIS XRI TC

4108 F. Example Local Resolver Interface Definition

4109 Following is a non-normative language-neutral example interface definition for a XRI resolver
4110 consistent with the requirements of this specification.

4111 The interface definition is provided as five operations where each operation takes two or more of
4112 the following input parameters. These input parameters correspond to the normative text in
4113 section 8.1. In all of these parameters, the value empty string ("") is interpreted the same as the
4114 value null.

4115

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.

4116

4117 The flags parameter is a binding-specific container data structure that encapsulates the
4118 following subparameters of the Resolution Output Format parameter. All of these are Boolean
4119 parameters defined in Table 6 in section 3.3.

4120

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.

4121

4122 Note that one subparameter defined in in Table 6, sep (service endpoint), is not included in this
4123 flags table because it is implicitly represented in the operation being called. The five operations
4124 shown in the table below correspond to the five possible combinations of the value of the
4125 Resolution Output Format parameter and the sep subparameter. (Note that if the Resolution
4126 Output Format is URI List, the sep subparameter MUST be considered to be TRUE, so there is
4127 no resolveAuthToURIList operation.)

4128

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

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

4130 **1. Resolve Authority to XRDS**

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

4133 • Performs authority resolution only (sections 9 and 10) and outputs an XRDS document as
4134   specified in section 8.2.1 when the `sep` subparameter is FALSE.
4135 • Only the authority component of the QXRI is processed by this function. If the QXRI contains
4136   a path or query component, it is ignored.
4137 • Returns a binding-specific representation of the resolution result which may include, but is not
4138   limited to, XRDS output, success/failure code, exceptions and error context.
4139 • The XRD element(s) in the output XRDS will be signed or not depending on the value of the
4140   `saml` flag.

4141

4142 **2. Resolve Authority to XRD**

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

4145 • Performs authority resolution only (sections 9 and 10) and outputs an XRD element as
4146   specified in section 8.2.2 when the `sep` subparameter is FALSE.
4147 • Only the authority component of the QXRI is processed by this function. If the QXRI contains
4148   a path or query component, it is ignored.
4149 • Returns a binding-specific representation of the resolution result which may include, but is not
4150   limited to, XRD output, success/failure code, exceptions and error context.
4151 • The output XRD will be signed or not depending on the value of the `saml` flag.

4152

4153

3. Resolve Service Endpoint to XRDS

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

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

4. Resolve Service Endpoint to XRD

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

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

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

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

4190 • Performs authority resolution (sections 9 and 10) and service endpoint selection (section 13)
4191 and outputs a non-empty URI List or an error as specified in section 8.2.3.

4192 • Returns a binding-specific representation of the resolution result which may include, but not
4193 limited to, URI-list output, success/failure code, exceptions and error context.

4194 • If successful, the output URI-list will contain zero or more elements. It is possible that the
4195 selected service contains no URI element and it is up to the consuming application to
4196 interpret such a result.

4197

G. Revision History

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

- http://www.oasis-open.org/committees/download.php/11853

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

- http://www.oasis-open.org/committees/download.php/17293

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

- http://wiki.oasis-open.org/xri/Xri2Cd02/ResWorkingDraft11

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

- http://www.oasis-open.org/committees/download.php/26277

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

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

- The reference to the XRI Syntax 2.0 specification in section 1.5 was updated.
- The XRD elements in sections 4.2.1 – 4.2.6 were reformatted to include attribute definitions as separate bullet points (per comment received from Eran Hammer-Lahav).
- The `xrd:XRD/xrd:Type` element was added to the XRD schema (section 4.2.1 and Appendix B and C) to reuse the `xrd:XRD/xrd:Service/xrd:Type` element at the XRD level in order to support extension specifications (per comment received from Eran Hammer-Lahav). A reference to this change was added in section 17.1.1
- The flowcharts in Figures 5, 6, 7, and 8 were edited for improved clarity about recording XRDs and nested XRDS documents and clarify using a Redirect URI as an input.
- The Next Authority URI construction algorithm in section 9.1.10 was revised slightly to accommodate using query strings.
- The wording of the bullet points in section 12.1 were clarified (per comment received from Eran Hammer-Lahav).
- A fourth example was added in section 12.5.1 to illustrate double XRDS nesting.
- Clarifications were made to the pseudocode in section 13.6.
- The CanonicalID verification rule for XRIs was simplified to eliminate the need to involve the `xrd:XRD/xrd:ProviderID` element (per suggestion from editor William Tan).
- Several typos and incorrect internal references were fixed.
- Several errors were fixed in the RNC schema.