# Extensible Resource Identifier (XRI) Resolution Version 2.0

## Working Draft 11, ED 08

## 7 November 2007

**Specification URIs:**

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

**This Version:**

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

**Previous Version:**

N/A

**Latest Version:**

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

**Latest Approved Version:**

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

**Technical Committee:**

OASIS eXtensible Resource Identifier (XRI) TC

**Chairs:**

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

**Editors:**

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

**Related Work:**

**Comment [DSR2]:** PROOF – confirm these with Mary McRae

This specification replaces or supercedes:

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

This specification is related to:

- Extensible Resource Identifier (XRI) Syntax Version 2.0, Committee Specification, December 2005
- Extensible Resource Identifier (XRI) Metadata Version 1.0,  Committee Draft 01, March 2005

## 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

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

## 1.1 Overview of XRI Resolution Architecture

Resolution is the function of dereferencing an identifier to a set of metadata describing the identified resource. For example, in DNS, a domain name is typically resolved using the UDP protocol into the IP address or other attributes of an Internet host. A federated domain name such as `docs.oasis-open.org` is resolved recursively from right to left, i.e., first the resolver queries the `org` nameserver for the IP address of the name-server for `oasis-open`, then it queries the `oasis-open` nameserver for the IP address for `docs`.

Non-recursing resolvers rely on *recursing nameservers* to do this work. For example, a non-recursing resolver might query a recursing nameserver for the entire DNS name `docs.oasis-open.org`. The nameserver would then do the job of querying the `org` nameserver for the IP address of `oasis-open`, then the `oasis-open` nameserver or the IP address of `docs`, and then return the result to the resolver. A recursing nameserver typically caches all these resource records so it can answer subsequent queries directly from cache.

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

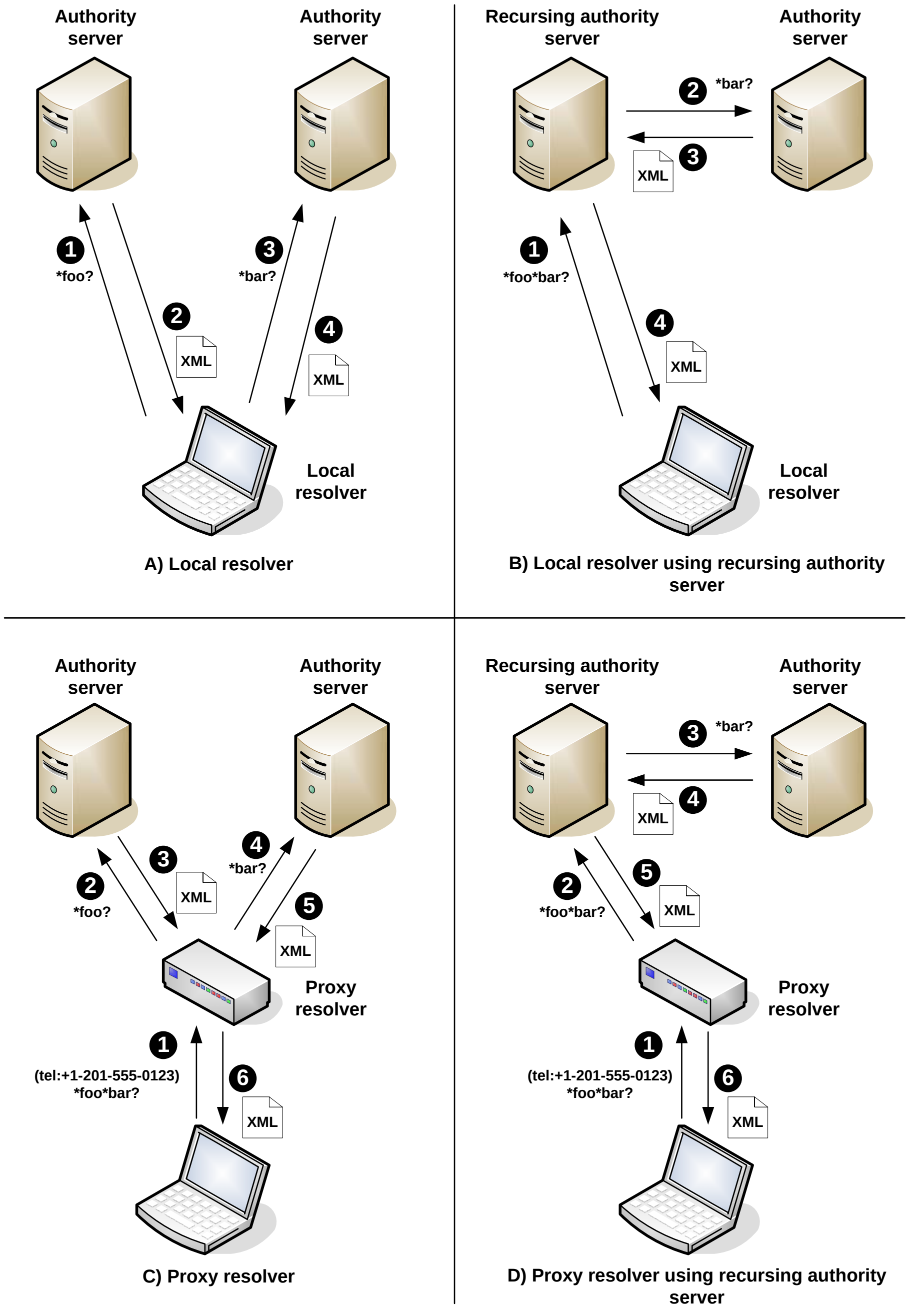
| Resolution Component | DNS Architecture | XRI Architecture |
|---|---|---|
| Identifier | domain name | XRI (authority + path + query) |
| Resource record format | text (resource record) | XML (XRDS document) |
| Attribute identifier | string | anyURI |
| Network endpoint identifier | IP address |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

| | | |
|---|---|---|
| | | resolver |

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

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

Figure 1 shows four scenarios of how these components might interact to resolve `xri://(tel:+1-201-555-0123)*foo*bar` (note that, 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**

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

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

- *Phase 1: Authority Resolution.* This is the phase required to resolve the authority segment of an XRI into an XRDS document describing the target authority. Authority resolution works iteratively from left-to-right across each subsegment in the authority segment of the XRI. in XRIs, subsegments are delimited using either a specified set of symbol characters or parentheses. For example, in the XRI `xri://(tel:+1-201-555-0123)*foo*bar`, the authority subsegments are `(tel:+1-201-555-0123)` (the community root authority, in this case a URI expressed as an cross-reference delimited with parentheses), `*foo`, (the first resolvable subsegment), and `*bar`, (the second resolvable subsegment). Note that a resolver must be preconfigured (or have its own way of discovering) the community root authority starting point, so the community root subsegment is not resolved except in one special case (see section 9.1.6).
- *Phase 2: Service Endpoint Selection.* Once authority resolution is complete, the optional second phase of XRI resolution is to select a specific set of metadata from the final XRDS document retrieved. Although an XRDS document may contain any type of metadata describing the target resource, this specification defines a ruleset for selecting *service endpoints*: descriptors of concrete URIs at which network services are available for the target resource. An XRI resolver may optionally use the path and/or query components of an XRI to select the service endpoint(s) to return to a consuming application.

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

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

## 1.2 Structure of this Specification

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 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 how to obtain XRDS metadata describing a resource, including synonyms for that resource, starting from an HTTP(S) URI identifying the resource.

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

- *XRI Resolution Flow* (section 7) provides a flowchart of the overall XRI resolution function.
- *Inputs and Outputs* (section 8) specifies the standard input parameters, output formats, and processing rules.
- *Generic Authority Resolution* (section 9) specifies a simple resolution protocol for the authority segment 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 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 and wikis 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. Mealling, 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.

**[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.

**[RFC4288]** N. Freed, J. Klensin, *Media Type Specifications and Registration Procedures*, http://www.ietf.org/rfc/rfc4288.txt, IETF RFC 4288, December 2005.

**[SAML]** S. Cantor, J. Kemp, R. Philpott, E. Maler, *Assertions and Protocols for the OASIS Security Assertion Markup Language* (SAML) V2.0, http://www.oasis-open.org/committees/security, March 2005.

**[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.

**[UUID]** Open Systems Interconnection – *Remote Procedure Call*, ISO/IEC 11578:1996, http://www.iso.org/, August 2001.

**[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.

**[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.

**[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.

**[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.

**[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.

**[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.

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

## 1.6 Non-Normative References

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

**[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/apps/org/workgroup/xri/download.php/2523/xri-requirements-and-glossary-v1.0.doc, June 2003.

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

**[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 provided a limited subset of the functionality of XRI resolvers.
2. *XRDS servers*, which provided 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 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.

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

## 2.7 XRI Authority Servers

### 2.7.1 Generic

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

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

### 2.7.2 HTTPS

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

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

### 2.7.3 SAML

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

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

## 2.8 Extensions

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

## 2.9 Language

This specification’s normative language is English. Translation into other languages is encouraged.

# 3 Namespaces

## 3.1 XRI Namespaces for XRI Resolution

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

### 3.1.1 XRIs Reserved for XRI Resolution

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

| **XRI<br>(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 |

*Table 2: XRIs reserved for XRI resolution.*

### 3.1.2 XRIs Assigned to XRI Resolution Service Types

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

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

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

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

## 3.2 XML Namespaces for XRI Resolution

Throughout this document, the following XML namespaces 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 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 C |
| application/xrd+xml | Content type for returning only the final XRD descriptor in a resolution chain | Appendix D |
| text/uri-list | Content type for returning a list of URIs output from the service endpoint selection process defined in section 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. 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.

| Media Type Parameter | Values | Usage | See Section |
|---|---|---|---|
| https | true or 1<br>false or 0 | Specifies use of HTTPS trusted resolution | 10.1 |
| saml | true or 1<br>false or 0 | Specifies use of SAML trusted resolution | 10.2 |
| refs | true or 1<br>false or 0 | Specifies whether Refs should be followed during resolution (by default they are followed) | 12.4 |
| sep | true or 1<br>false or 0 | Specifies whether service endpoint selection should be performed | 13 |

|  |  |  |  |
|---|---|---|---|
| nodefault_t | true or 1<br>false or 0 | Specifies whether a default match on a Type service endpoint selection element is allowed | 13.3 |
| nodefault_p | true or 1<br>false or 0 | Specifies whether a default match on a Path service endpoint selection element is allowed | 13.3 |
| nodefault_m | true or 1<br>false or 0 | Specifies whether a default match on a MediaType service endpoint selection element is allowed | 13.3 |
| uric | true or 1<br>false or 0 | Specifies whether a resolver should automatically construct service endpoint URIs | 13.7.1 |
| cid | true or 1<br>false or 0 | Specifies whether automatic canonical ID verification should performed (by default it is performed) | 14.3 |

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

See sections 8 - 14 for more about usage of these media types and media type parameters.

# 4 XRDS Documents

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

## 4.1 XRDS and XRD Namespaces

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

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

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

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

## 4.2 XRD Elements and Attributes

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

```
<XRDS xmlns="xri://$xrds" ref="xri://(tel:+1-201-555-0123)*foo">
    <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
        <Query>*foo</Query>
        <Status code="100"/>
        <ServerStatus code="100"/>
        <Expires>2005-05-30T09:30:10Z</Expires>
        <ProviderID>
         xri://(tel:+1-201-555-0123)
        </ProviderID>
        <LocalID>*baz</LocalID>
        <EquivID>https://example.com/example/resource/</EquivID>
        <CanonicalID>xri://(tel:+1-201-555-0123)!1234</CanonicalID>
        <CanonicalEquivID>
         xri://=!4a76!c2f7!9033.78bd
        </CanonicalEquivID>
        <Service>
            <ProviderID>
             xri://(tel:+1-201-555-0123)!1234
            </ProviderID>
            <Type>xri://$res*auth*($v*2.0)</Type>
            <MediaType>application/xrds+xml</MediaType>
            <URI priority=”10”>http://resolve.example.com</URI>
            <URI priority=”15”>http://resolve2.example.com</URI>
            <URI>https://resolve.example.com</URI>
        </Service>
        <Service>
            <ProviderID>
             xri://(tel:+1-201-555-0123)!1234
            </ProviderID>
            <Type>xri://$res*auth*($v*2.0)</Type>
            <MediaType>application/xrds+xml;https=true</MediaType>
            <URI>https://resolve.example.com</URI>
        </Service>
        <Service>
            <Type match="null" />
            <Path select="true">media/pictures</Path>
            <MediaType select="true">image/jpeg</MediaType>
            <URI append="path" >http://pictures.example.com</URI>
        </Service>
        <Service>
            <Type match="null" />
            <Path select="true">media/videos</Path>
            <MediaType select="true">video/mpeg</MediaType>
            <URI append="path" >http://videos.example.com</URI>
        </Service>
        <Service>
            <ProviderID> xri://!!1000!1234.5678</ProviderID>
            <Type match="null" />
            <Path match="default" />
            <URI>http://example.com/local</URI>
        </Service>
        <Service>
            <Type>http://example.com/some/service/v3.1</Type>
            <URI>http://example.com/some/service/endpoint</URI>
        </Service>
    </XRD>
</XRDS>
```

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

### 4.2.1 Management Elements

The first set of elements are used to manage XRDs, particularly from the perspective of caching and error handling. 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. Includes an OPTIONAL `xml:id` attribute of type `xs:ID`. This attribute is REQUIRED in trusted resolution to uniquely identify this element within the containing `xrds:XRDS` document. It also includes an OPTIONAL `idref` attribute of type `xs:idref`. This attribute is REQUIRED in trusted resolution when an XRD element in a nested `xrd:XRDS` document must reference a previously included XRD instance. See sections 4.3 and 12.1. Lastly, it includes a `version` attribute that is OPTIONAL for uses outside of XRI resolution but REQUIRED for XRI resolution as defined in section 4.3.2

**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. REQUIRED if the resolver must report certain error conditions. Contains a REQUIRED attribute `code` of type `xs:int` that provides a numeric status code. Contains OPTIONAL enumerated attributes `cid` and `ceid` that report the results of CanonicalID verification as defined in section 14.3.4. 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 and `code` attribute are defined in section 15.

**xrd:XRD:xrdServerStatus**

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

**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. Accepts the optional global `priority` attribute (section 4.3.3). 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.

**xrd:XRD/xrd:Ref**

0 or more more per `xrd:XRD` element. Type `xs:anyURI`. MUST contain an absolute XRI. Accepts the optional global `priority` attribute (section 4.3.3). Choice between this or the `xrd:XRD/xrd:Redirect` element above. MUST be processed by a resolver

(depending on the value of the `refs` media type parameter) to locate another XRDS document authorized to describe the target resource as defined in section 12.

### 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`. 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. For purposes of CanonicalID verification (section 14.3), 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 resolution service endpoint (see below).

**xrd:XRD/saml:Assertion**

0 or 1 per `xrd:XRD`. A SAML assertion from the parent authority (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`. Accepts the optional global `xrd:priority` attribute (section 4.3.3). Asserts a interchangeable synonym for the value of the `xrd:Query` element. See section 5.2.1. MUST be assigned by the same parent authority providing the current XRD. Does not require verification.

**xrd:XRD/xrd:EquivID**

0 or more per `xrd:XRD` element. Type `xs:anyURI`. Accepts the optional global `priority` attribute (section 4.3.3). Asserts an absolute synonym for the query identifier that is not equivalent to the CanonicalID or CanonicalEquivID (see below). See section 5.2.2. MUST be an absolute identifier. MAY be verified as defined in section 14.2. MAY be used for CanonicalEquivID verification as defined in section 14.3.3.

**xrd:XRD/xrd:CanonicalID**

0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical synonym for the query identifier assigned by the identifier authority issuing the XRD. See section 5.2.3. MUST be an absolute identifier. SHOULD be verified as defined in section 14.3.

**xrd:XRD/xrd:CanonicalEquivID**

0 or 1 per `xrd:XRD` element. Type `xs:anyURI`. Asserts the canonical synonym for the query assigned by any identifier authority. See section 5.2.4. MUST be an absolute identifier. SHOULD be verified as defined in section 14.3.3.

### 4.2.4 Service Endpoint Descriptor Elements

The next set of elements are 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 described 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 (see section 4.3.3). Note that to prevent processing conflicts, the XRD schema permits only one of these element types in a service endpoint: `xrd:URI`, `xrd:Redirect`, or `xrd:Ref`.

**xrd:XRD/xrd:Service**

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

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

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

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

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

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

0 more per `xrd:XRD/xrd:Service` element. Choice between this or the `xrd:XRD/xrd:Service/xrd:URI` or `xrd:XRD/xrd:Service/xrd:Ref` elements. Type `xs:anyURI`. Identical to the `xrd:XRD/xrd:Redirect` element defined above except processed only in the context of service endpoint selection. See section 12. Includes an optional `append` attribute that governs construction of the final redirect URI as defined in section 13.7.

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

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

### 4.2.5 Service Endpoint Trust Elements

Similar to the 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 current XRD. In XRI resolution, this means 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 are used in XRI resolution to select service endpoints. They include two global attributes used for this purpose: `match` and `select`. See sections 13.3.2 and 13.4.2.

**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.

**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.

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

> 0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. The media type of content available at this service endpoint. The value of this element MUST be of the form of a media type defined in **[RFC2046]**. See section 3.3 for the media types used in XRI resolution. See section 13.3.8 for MediaType element matching rules.

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

## 4.3 XRD Attribute Processing Rules

### 4.3.1 ID Attribute

For uses such as SAML trusted resolution (section 10.2) that require unique identification of multiple XRD elements within an XRDS document, the XRD element uses an optional `xml:id` attribute as defined by the W3C XML ID specification **[XMLID]**. If present, the value of this element MUST be unique for all elements in the containing XML document. Because an XRI resolver may need to assemble multiple XRDs received from different authority servers into one XRDS document, there MUST be negligible probability that the value of the `xrd:XRD/@xml:id` attribute is not globally unique. For this reason the value of this attribute SHOULD be a UUID as defined by **[UUID]** prefixed by a single underscore character (“_”) in order to make it a legal *NCName* as required by **[XMLID]**. However the value of this attribute MAY be generated by any algorithm that fulfills the same requirements of global uniqueness and *NCName* conformance.

Note that when an XRI resolver is assembling multiple XRDs into a single XRDS document, their XML document order MUST match the order in which they were resolved (see section 9.1.2). Also, if reference processing requires the same XRD to be included in an XRDS document twice (via a nested XRDS document), that XRD MUST reference the previous instance using the `xrd:XRD/@xml:idref` attribute as defined in section 12.

### 4.3.2 Version Attribute

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

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

### 4.3.3 Priority Attribute

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

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

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

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

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

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

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

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

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

# 5 XRD Synonym Elements

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

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

*Table 7: The four XRD synonym elements.*

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

## 5.1 Query Identifiers

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

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

### 5.1.1 HTTP(S) URI Query Identifiers

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

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

### 5.1.2 XRI Query Identifiers

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

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

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

## 5.2 Synonym Elements

### 5.2.1 LocalID

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

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

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

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

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

### 5.2.2 EquivID

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

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

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

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

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

### 5.2.3 CanonicalID

The purpose of the `xrd:CanonicalID` element is to assert the parent authority's own canonical synonym for the current fully-qualified query identifier. A CanonicalID MUST meet all the requirements of an EquivID plus the following:

1. It MUST be an identifier for which the parent authority is the final authority. This means it MUST resolve to the same XRD as the current fully-qualified query identifier, i.e., an XRD containing the same elements and values (with the exception of the values of the `xrd:XRD/xrd:Query`, `xrd:XRD/xrd:Expires`, and `xrd:XRD/xrd:LocalID` elements).
2. If it is any XRI except a community root authority XRI (section 9.1.3), it MUST be a direct child of the parent authority's own CanonicalID. For example, if the CanonicalID asserted for a target resource is `@!1!2!3`, then the CanonicalID for the parent authority issuing the XRD must be `@!1!2`.
3. Once assigned, a parent authority SHOULD NEVER: a) change or reassign a CanonicalID value, or b) stop asserting a CanonicalID element in an XRD in which it has been asserted. For this reason, it is STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI **[XRISyntax]** or a URN **[RFC2141]**.

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

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

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

### 5.2.4 CanonicalEquivID

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

1. In order for the value of the `xrd:CanonicalEquivID` element to be verified: a) the XRD in which it appears MUST include a CanonicalID that can be verified as specified in section 14.2, and b) the XRD to which it resolves MUST include an EquivID backpointer to this CanonicalID as specified in section 14.3.3.
2. For the same reasons as with a CanonicalID, it is STRONGLY RECOMMENDED to use a persistent identifier such as a persistent XRI **[XRISyntax]** or a URN **[RFC2141]**.

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

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

## 5.3 Redirect and Ref Elements

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

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

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

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

## 5.4 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.

- EquivID verification is not performed automatically by XRI resolvers but may be easily performed by consuming applications using one additional XRI resolution call as specified in section 14.2.
- CanonicalID and CanonicalEquivID verification are performed automatically by XRI resolvers (unless this function is explicitly turned off) as specified in section 14.3. The `cid` and `ceid` attributes of the `xrd:XRD/xrd:Status` element report whether the CanonicalID and CanonicalEquivID were present and verified as specified in section 14.3.4.

## 5.5 Synonym Selection

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

- Only select a verified synonym (see above).
- Select a persistent synonym, particuarly 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 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

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

## 6.1 Overview

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

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

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

## 6.2 HEAD Protocol

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

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

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

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

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

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

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

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

## 6.3 GET Protocol

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

The XRDS server response MUST be one of four options:

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

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

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

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

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

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

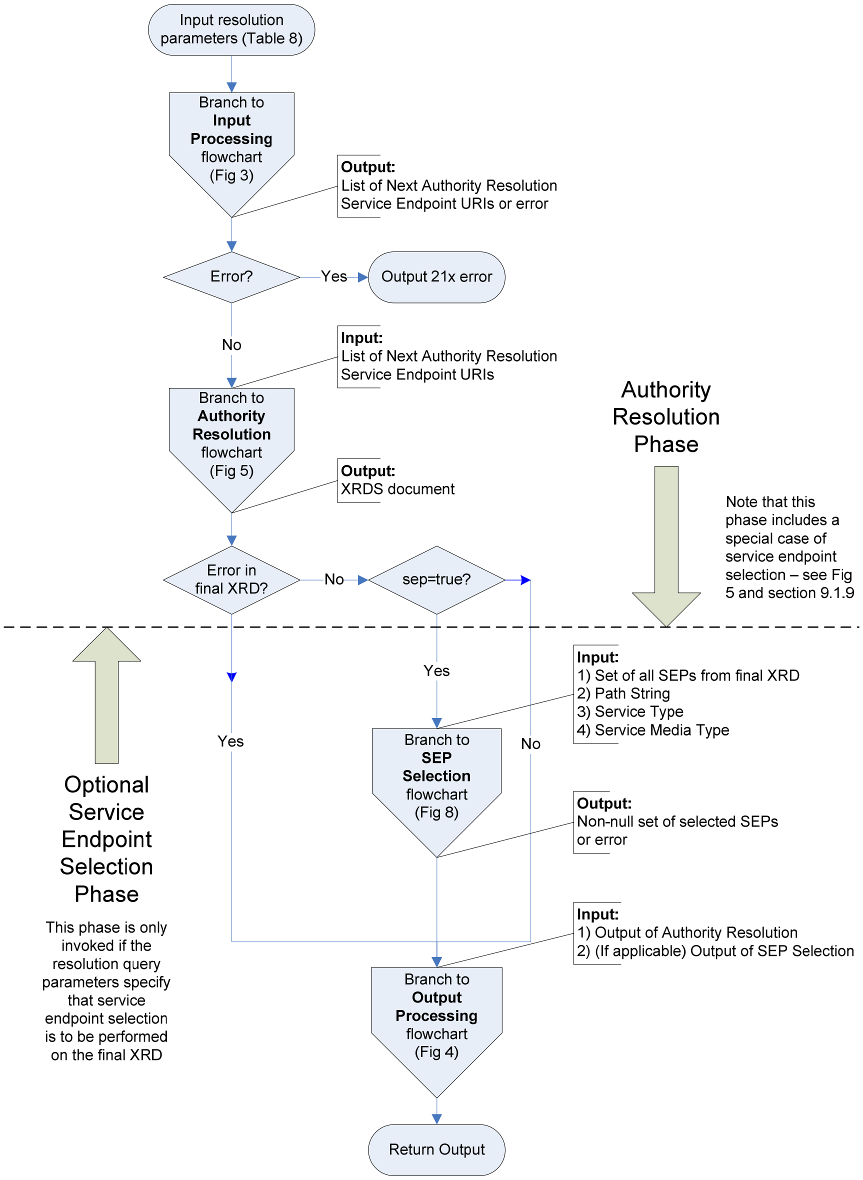
In all cases the HTTP(S) status messages and error codes defined in **[RFC2616]** apply.

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

**Comment [DSR3]:** PROOF – New text was added to this section.

# 7 XRI Resolution Flow

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



Input resolution parameters (Table 8)
Branch to **Input Processing** flowchart (Fig 3)
**Output:** List of Next Authority Resolution Service Endpoint URIs or error
Error?
Yes
Output 21x error
No
**Input:** List of Next Authority Resolution Service Endpoint URIs
Branch to **Authority Resolution** flowchart (Fig 5)
**Output:** XRDS document
Authority Resolution Phase
Note that this phase includes a special case of service endpoint selection – see Fig 5 and section 9.1.9
Error in final XRD?
No
sep=true?
Yes
Yes
No
**Input:** 1) Set of all SEPs from final XRD 2) Path String 3) Service Type 4) Service Media Type
Branch to **SEP Selection** flowchart (Fig 8)
**Output:** Non-null set of selected SEPs or error
Optional Service Endpoint Selection Phase
This phase is only invoked if the resolution query parameters specify that service endpoint selection is to be performed on the final XRD
**Input:** 1) Output of Authority Resolution 2) (If applicable) Output of SEP Selection
Branch to **Output Processing** flowchart (Fig 4)
Return Output

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

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

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

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

# 8 Inputs and Outputs

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

## 8.1 Inputs

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

| Logical Input Parameter Name | Type | Required/ Optional | Default | Resolution Phase | Section |
|---|---|---|---|---|---|
| QXRI (query XRI) including Authority String, Path String, and Query String | xs:anyURI | Required | N/A | Authority Resolution (except Path String which is used in Service Endpoint Selection) | 8.1.1 |
| Resolution Output Format | xs:string (media type) | Optional | Null | Authority Resolution | 8.1.2 |
| Service Type | xs:anyURI | Optional | Null | Service Endpoint Selection | 8.2.3 |
| Service Media Type | xs:string (media type) | Optional | Null | Service Endpoint Selection | 8.1.4 |

*Table 8: Input parameters for XRI resolution.*

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

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

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

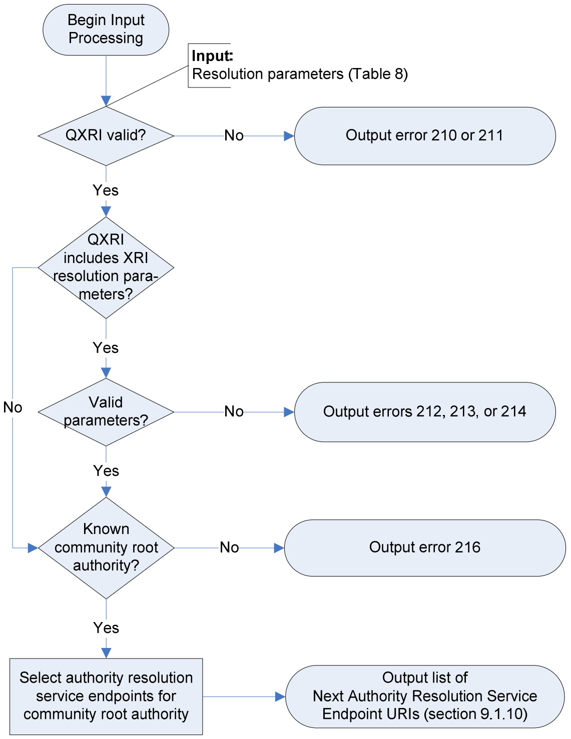

Begin Input Processing
Input:
Resolution parameters (Table 8)
QXRI valid?
No
Output error 210 or 211
Yes
QXRI includes XRI resolution para-meters?
Yes
No
Valid parameters?
No
Output errors 212, 213, or 214
Yes
Known community root authority?
No
Output error 216
Yes
Select authority resolution service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

*Figure 3: Input processing flowchart.*

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

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

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

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

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

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

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

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

### 8.1.2 Resolution Output Format

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

- The media type for the resolution response.
- Whether generic or trusted resolution must be used by the resolver.
- Whether Refs should be followed during resolution.
- Whether CanonicalID verification should not be performed during resolution.
- Whether service endpoint selection should be performed on the final XRD.
- Whether default matches should be ignored during service endpoint selection.

- Whether URIs should automatically be constructed in the final XRD.

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

1. The value of Resolution Output Format MUST be one of the values specified in Table 5 and MAY include any of the media type parameters specified in Table 6.
2. If the value of the `https` media type parameter is TRUE, the resolver MUST use the HTTPS trusted authority resolution protocol specified in section 10.1 (or return an error indicating this is not supported).
3. If the value of the `saml` media type parameter is TRUE, the resolver MUST use the SAML trusted authority resolution protocol specified in section 10.2 (or return an error indicating this is not supported).
4. If the value of both the `https` and `saml` media type parameters are TRUE, the resolver MUST use the HTTPS+SAML trusted authority resolution protocol specified in section 10.3 (or return an error indicating this is not supported).
5. If the value of the `cid` media type parameter is TRUE or null, or if the parameter is absent, the resolver MUST perform CanonicalID verification as specified in section 14.3. If the value of the `cid` media type parameter is FALSE, the resolver MUST NOT perform CanonicalID verification.
6. If the value of the `refs` media type parameter is TRUE or null, or if the parameter is absent, the resolver MUST perform Ref processing as specified in section 12. If the value of the `refs` media type parameter is FALSE, the resolver MUST NOT perform Ref processing and must return an error if a Ref is encountered as specified in section 12.
7. If the value of the `sep` media type parameter is TRUE, the resolver MUST perform service endpoint selection on the final XRD. If the value of the `sep` media type parameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT perform service endpoint selection on the final XRD unless it is required to produce a URI List or HTTP(S) redirect. See section 8.2.
8. If the value of the `nodefault_r`, `nodefault_p`, or `nodefault_m` media media type parameter is TRUE, the resolver MUST ignore default matches on the corresponding service endpoint selection element categories as specified in section 13.3.2.
9. If the value of the `uric` media type parameter is TRUE, the resolver MUST perform service endpoint URI construction as specified in section 13.7.1. If the value of the `uric` media type parameter is FALSE or null, or if the parameter is absent, the resolver MUST NOT perform service endpoint URI construction.

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

### 8.1.3 Service Type

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

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

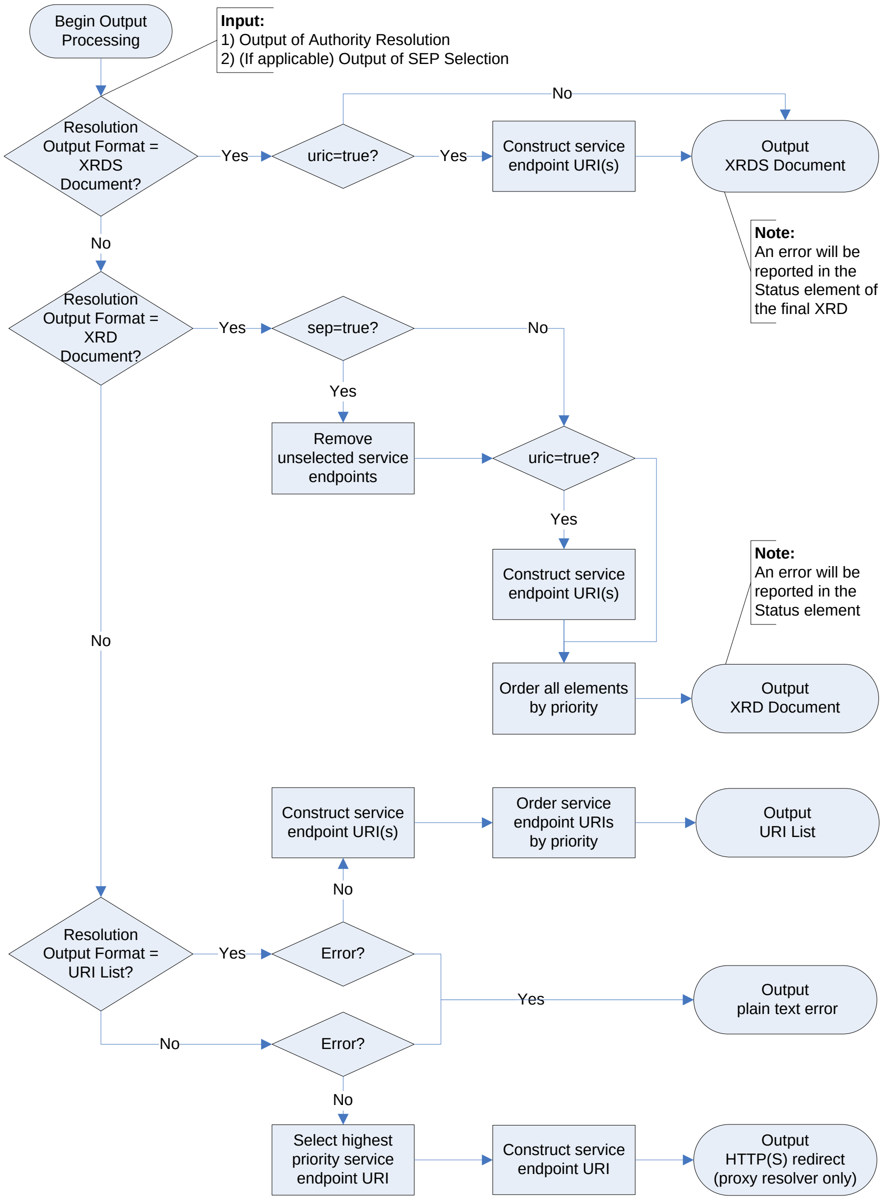

Begin Output Processing
Input:
1) Output of Authority Resolution
2) (If applicable) Output of SEP Selection
Resolution Output Format = XRDS Document?
Yes
uric=true?
Yes
No
Construct service endpoint URI(s)
Output XRDS Document
Note:
An error will be reported in the Status element of the final XRD
No
Resolution Output Format = XRD Document?
Yes
sep=true?
No
Yes
Remove unselected service endpoints
uric=true?
Yes
Construct service endpoint URI(s)
Note:
An error will be reported in the Status element
Order all elements by priority
Output XRD Document
No
Construct service endpoint URI(s)
Order service endpoint URIs by priority
Output URI List
No
Resolution Output Format = URI List?
Yes
Error?
Yes
Output plain text error
No
Error?
No
Select highest priority service endpoint URI
Construct service endpoint URI
Output HTTP(S) redirect (proxy resolver only)

*Figure 4: Output processing flowchart.*

The following sections provide additional construction and validation requirements.

### 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 A.
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` parameter 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` media type parameter 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` media type parameter is TRUE, synonym verification MUST be resported 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 A.
2. The XRD elements MUST conform to the additional requirements in section 4.
3. If the value of the `saml` parameter 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` media type parameter 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` media type parameter 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, Service Media Type, and Path String) are null.
6. If service endpoint selection is performed, the only `xrd:Service` elements in the XRD element MUST be those selected according to the rules specified in section 13. If no service endpoints were selected by those rules, no `xrd:Service` elements will be present. In addition, all elements within the XRD element that are subject to the global

`priority` attribute (even if the attribute is absent or null) MUST be returned in order of highest to lowest priority as defined in section 4.3.3. *Any other filtering of the XRD element MUST NOT be performed.* Note that this means that if the XRD element includes a SAML signature element as defined in section 10.2, this element is still returned inside the XRD element even though it may not be able to be verified by a consuming application.

7. If the value of the `cid` media type parameter is TRUE, synonym verification MUST be resported using the `xrd:Status` element of each XRD in the XRDS document as defined in section 14.
8. If the output is an error, this error MUST be returned using the `xrd:Status` element as defined in section 15.

### 8.2.3 URI List

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

1. For this output, service endpoint selection is REQUIRED, even if the values of all three service endpoint selection input parameters (Service Type, Service Media Type, and Path String) are null.
2. If authority resolution and service endpoint selection are both successful, the output MUST be a valid URI List as defined by section 5 of **[RFC2483]**.
3. If, after applying the service endpoint selection rules, more than one service endpoint is selected, the highest priority `xrd:XRD/xrd:Service` element MUST be selected as defined in section 4.3.3.
4. If the final selected `xrd:XRD/xrd:Service` element contains a `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref` element, Redirect and Ref processing MUST be performed as described in section 12.
5. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s) MUST be constructed as defined in section 13.7.
6. The URIs MUST be returned in order of highest to lowest priority of the source `xrd:URI` elements within the selected `xrd:Service` element as defined in section 4.3.3. When two or more of the source `xrd:URI` elements have equal priority, their constructed URIs SHOULD be returned in random order. *Any other filtering of the URI list MUST NOT be performed.*
7. If the output is an error, it MUST be returned with the content type `text/plain` as defined in section 15.

### 8.2.4 HTTP(S) Redirect

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

# 9 Generic Authority Resolution Service

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

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

## 9.1 XRI Authority Resolution

### 9.1.1 Service Type and Service Media Type

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

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

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

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

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

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

### 9.1.2 Protocol

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

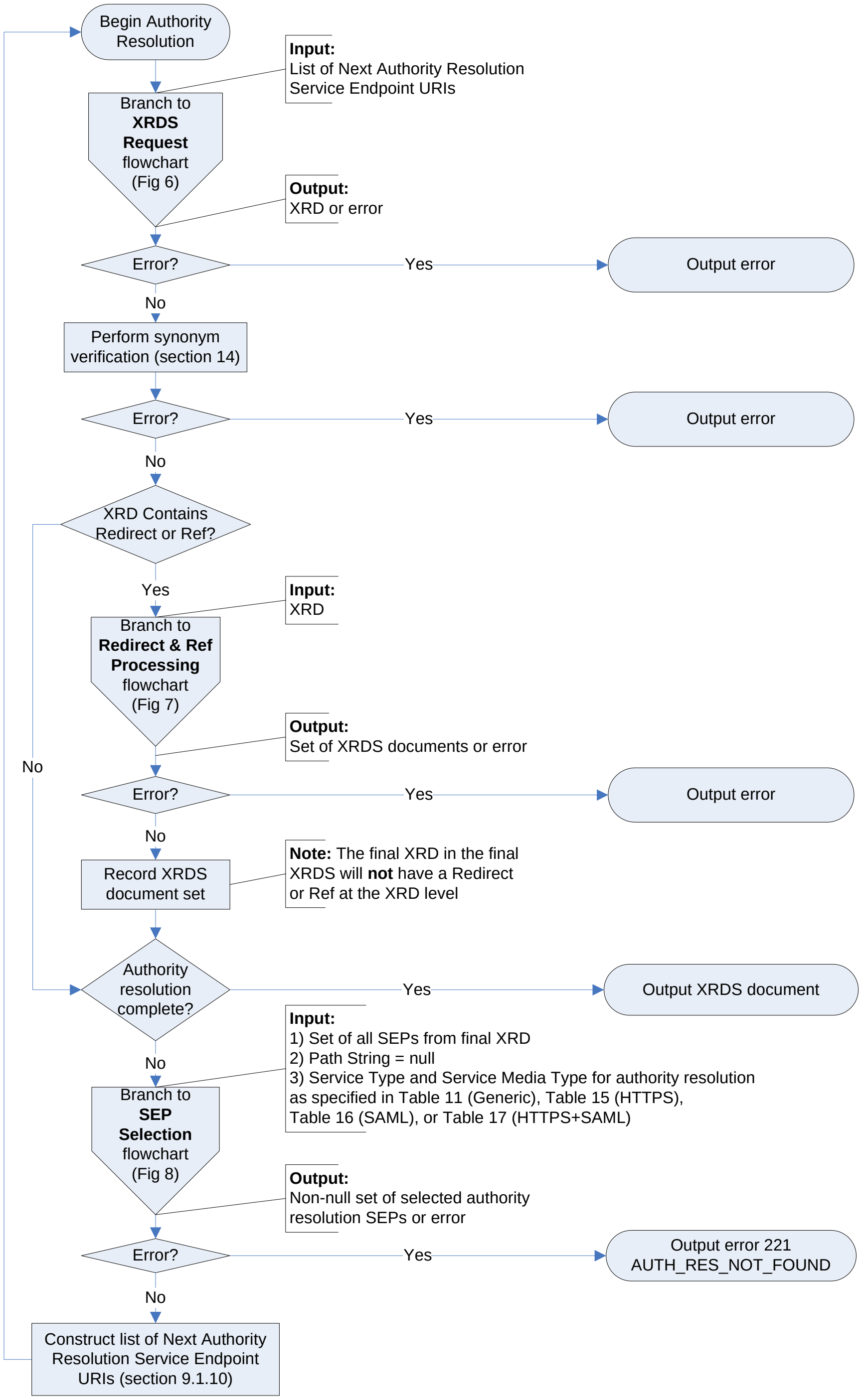

Begin Authority Resolution
Input:
List of Next Authority Resolution Service Endpoint URIs
Branch to **XRDS Request** flowchart (Fig 6)
Output:
XRD or error
Error?
Yes
Output error
No
Perform synonym verification (section 14)
Error?
Yes
Output error
No
XRD Contains Redirect or Ref?
Yes
No
Input:
XRD
Branch to **Redirect & Ref Processing** flowchart (Fig 7)
Output:
Set of XRDS documents or error
Error?
Yes
Output error
No
Record XRDS document set
**Note:** The final XRD in the final XRDS will **not** have a Redirect or Ref at the XRD level
Authority resolution complete?
Yes
Output XRDS document
No
Input:
1) Set of all SEPs from final XRD
2) Path String = null
3) Service Type and Service Media Type for authority resolution as specified in Table 11 (Generic), Table 15 (HTTPS), Table 16 (SAML), or Table 17 (HTTPS+SAML)
Branch to **SEP Selection** flowchart (Fig 8)
Output:
Non-null set of selected authority resolution SEPs or error
Error?
Yes
Output error 221 AUTH_RES_NOT_FOUND
No
Construct list of Next Authority Resolution Service Endpoint URIs (section 9.1.10)

*Figure 5: Authority resolution flowchart.*

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

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

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

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

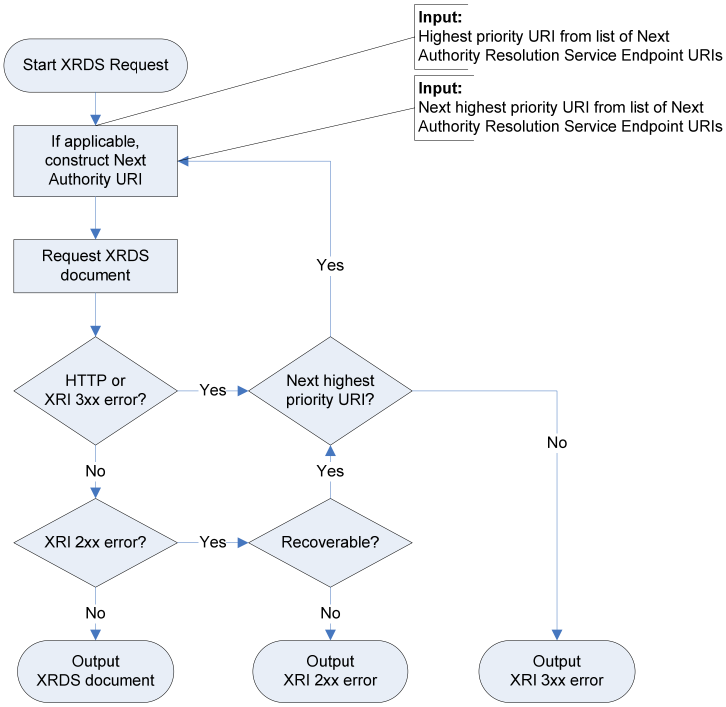

Input:
Highest priority URI from list of Next Authority Resolution Service Endpoint URIs
Input:
Next highest priority URI from list of Next Authority Resolution Service Endpoint URIs
Start XRDS Request
If applicable, construct Next Authority URI
Request XRDS document
Yes
HTTP or XRI 3xx error?
Yes
Next highest priority URI?
No
No
Yes
XRI 2xx error?
Yes
Recoverable?
No
No
Output XRDS document
Output XRI 2xx error
Output XRI 3xx error

*Figure 6: XRDS request flowchart.*

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 0.

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 media type parameters 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 15.4.

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 only effective 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.

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

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

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

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

### 9.1.5 Community Root Authorities

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

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

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

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

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

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

### 9.1.6 Self-Describing XRDS Documents

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

1. If the resolver knows an HTTP(S) URI for the target authority's XRI authority resolution service endpoint, it may use the resolution protocol specified in section 5 to request an XRDS document directly from this HTTP(S) URI(s). 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) the location of 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 a 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 a identifier authority had the community root authority identifier `xri://(example)` and 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 syntatic 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.

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

*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 |

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

### 9.1.8 Cross-References

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

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

| **Cross-reference type** | **Example XRI** | **Next Authority URI after resolving `xri://@!a!b`** |
|---|---|---|
| Absolute XRI | xri://@!a!b!(@!1!2!3)*e/f | http://example.com/xri-authority/!(@!1!2!3) |
| Absolute URI | xri://@!a!b*(mailto:jd@example.com)*e/f | http://example.com/xri-authority/*(mailto:jd@example.com) |
| Absolute XRI w/ XRI metadata | xri://@!a!b*($v/2.0)*e/f | http://example.com/xri-authority/*($v*2.0) |
| Relative XRI | xri://@!a!b*(c*d)*e/f | http://example.com/xri-authority/*(c*d) |
| Relative URI | xri://@!a!b*(foo/bar)*e/f | http://example.com/xri-authority/*(foo%2fbar) |

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

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

For each iteration of authority resolution, the resolver MUST select the next authority resolution service endpoint from the current XRD as specified in section 13. For generic authority resolution, this selection process MUST use the parameters specified in Table 11. For trusted authority resolution, this selection process MUST use the parameters specified in Table 15, Table 16, or Table 17. In all cases, an explicit match on the `xrd:XRD/xrd:Service/xrd:Type` element is

REQUIRED, so during authority resolution, a resolver MUST set the `nodefault` parameter to a value of `nodefault=type` in order to override selection of a default service endpoint as specified in section 13.3.2.

### 9.1.10 Construction of the Next Authority URI

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 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 to the path component of the Next Authority Service Endpoint URI. The resulting URI is called the *Next Authority URI*.

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

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

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

append uri-escape(next-auth-string) to path of next-auth-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).

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

## 9.2 IRI Authority Resolution

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

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

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

### 9.2.1 Service Type and Media Type

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

### 9.2.2 Protocol

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

1. The Next Authority URI (section 9.1.10) is constructed by extracting the entire IRI authority segment and prepending the string `http://`. See the exception in section 9.2.3.
2. The HTTP GET request MUST include an HTTP Accept header containing only the following:

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

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

```
Host: example.com
```

4. An HTTP server acting as an IRI authority SHOULD respond with an XRDS document containing the XRD describing that authority.
5. The responding server MUST use the value of the `Host:` header to populate the `xrd:XRD/xrd:Query` element in the resulting XRD. For example:

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

### 9.2.3 Optional Use of HTTPS

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

# 10 Trusted Authority Resolution Service

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 | Media Type Parameters |
|---|---|---|
| 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 resolver MUST output an 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"` parameter).
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 requests a content type of `application/xrds+xml; 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. The latter MAY be achieved by combining SAML trusted resolution with HTTPS trusted resolution as defined in section 10.3. Message confidentiality may also be achieved with other security protocols used in conjunction with this specification. SAML trusted resolution also does not provide a means for an authority to provide different responses for different requestors; client authentication is explicitly out-of-scope for version 2.0 of XRI resolution.

### 10.2.1 Service Type and Service Media Type

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

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

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

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

### 10.2.2 Protocol

#### 10.2.2.1 Client Requirements

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

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

#### 10.2.2.2 Server Requirements

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

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

2. The XRDS document returned by the resolution service MUST contain a `saml:Assertion` element as an immediate child of the `xrd:XRD` element that is valid per the processing rules described by [SAML].
3. The `saml:Assertion` element MUST contain a valid enveloped digital signature as defined by [XMLDSig] and as constrained by section 5.4 of [SAML].
4. The signature MUST apply to the `xrd:XRD` element that contains the signed SAML assertion. Specifically, the signature MUST contain a single `ds:SignedInfo/ds:Reference` element, and the `URI` attribute of this reference MUST refer to the `xrd:XRD` element that is the immediate parent of the signed SAML assertion. The URI reference MUST NOT be empty and it MUST refer to the identifier contained in the `xrd:XRD/@xml:id` attribute.
5. [SAML] specifies that the digital signature enveloped by the SAML assertion MAY contain a `ds:KeyInfo` element. If this element is included, it MUST describe the key used to verify the digital signature element. However, because the signing key is known in advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the `ds:Signature` element of the SAML assertion.
6. The `xrd:XRD/xrd:Query` element MUST be present, and the value of this field MUST match the XRI authority subsegment requested by the client.
7. The `xrd:XRD/xrd:ProviderID` element MUST be present and its value MUST match the value of the `xrd:XRD/xrd:Service/xrd:ProviderID` element in an XRD advertising availability of trusted resolution service from this authority as required in section 10.2.5.
8. The `xrd:XRD/saml:Assertion/saml:Subject/saml:NameID` element MUST be present and equal to the `xrd:XRD/xrd: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 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 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 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

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

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

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

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

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

## 10.3 HTTPS+SAML

### 10.3.1 Service Type and Service Media Type

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

| Service Type | Service Media Type | Media Type Parameters |
|---|---|---|
| xri://$res*auth*($v*2.0) | application/xrds+xml | https=true<br>saml=true |

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

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

### 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 that may implemented via a local resolver interface. This section defines a mapping of these functions to an HTTP(S) interface for remote invocation. This mapping is based on a standard syntax for expressing an XRI as an HTTP URI, called an *HXRI*, as defined in section 11.2. This includes a method of passing the other XRI resolution input parameters as query parameters in the HXRI.

Proxy resolution is useful for many reasons:

- 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 compatability 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 | Media Type Parameters |
|---|---|---|
| xri://$res*proxy*($v*2.0) | application/xrds+xml<br>application/xrd+xml<br>text/uri-list | All parameters 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 (including the optional media type parameters). 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.

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 media type 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:

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

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

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

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

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

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

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

```
proxy-reg-name    = "xri." ireg-name
QXRI              = xri-hier-part [ "?" i-query ]
```

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

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

## 11.3 HXRI Query Parameters

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

| Logical Parameter Name | HXRI Component | HXRI Query Parameter Name |
|---|---|---|
| QXRI | Entire path and query string of HXRI (exclusive of HXRI query parameters listed below) | N/A |
| Resolution Output Format | HXRI query parameter | _xrd_r |
| Service Type | HXRI query parameter | _xrd_t |
| Service Media Type | HXRI query parameter | _xrd_m |

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

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

1. The QXRI MUST be normalized as specified in section 11.2.
2. If the original QXRI has an existing query component, the HXRI query parameters MUST be appended to that query component. (Note that the query parameter names in Table 19 were chosen to minimize the probability of collision with any other existing query parameter names.) After proxy resolution, the HXRI query parameters MUST subsequently be removed from the QXRI query component. The existing QXRI query component MUST NOT be altered in any other way, i.e., it must be passed through with no changes in parameter order, escape encoding, etc.
3. If the original QXRI does not have a query component, one MUST be added to pass any HXRI query parameters. After proxy resolution, this query component MUST be entirely removed.
4. If the original QXRI had a null query component (only a leading question mark), or a query component consisting of only question marks, *one additional leading question mark* MUST be added before adding any HXRI query  parameters. After proxy resolution, any HXRI query parameters and exactly one leading question mark MUST be removed. See the URI construction steps defined in section 13.6.
5. Each HXRI query parameter MUST be delimited from other parameters by an ampersand (“&”).
6. Each HXRI query parameter MUST be delimited from its value by an equals sign (“=”).

7. If an HXRI query parameter includes one of the media type parameters defined in Table 6, it MUST be delimited from the HXRI query parameter with a semicolon (“`;`”).
8. If any HXRI query parameter name is included but its value is empty, the value of the parameter MUST be considered null.

## 11.4 HXRI Encoding/Decoding Rules

**Comment [DSR4]:** PROOF – text and example revised in ED08.

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

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

Once the HXRI is in URI-normal form, the following sequence of encoding steps MUST be performed in the order specified before an HXRI is submitted to a proxy resolver. *IMPORTANT: this sequence of steps is not idempotent, so it MUST be performed only once.*

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

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

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

| | |
|---|---|
| QXRI | `https://xri.example.com/=example*résumé/path?query` |
| _xrd_r | `_xrd_r=application/xrds+xml;https=true;sep=true` |
| _xrd_t | `_xrd_t=http://example.org/test?a=1&b=hello planéte` |
| _xrd_m | `_xrd_m=application/atom+xml` |

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

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

| | |
|---|---|
| QXRI | https://xri.example.com/=example*r%E9sum%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%20plan%E9te |
| _xrd_m | _xrd_m=application/atom+xml |

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

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

| | |
|---|---|
| QXRI | https://xri.example.com/=example*r%25E9sum%25E9/path?query |
| _xrd_r | _xrd_r=application/xrds+xml%3Bhttps=true%3Bsep=true |
| _xrd_t | _xrd_t=http://example.org/test?a=1%26b=hello%2520plan%25E9te |
| _xrd_m | _xrd_m=application/atom+xml |

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

Following is the fully-encoded HXRI:

```
https://xri.example.com/=example*r%25E9sum%25E9/path?query
&_xrd_r=application/xrds+xml%3Bhttps=true%3Bsep=true
&_xrd_t=http://example.org/test?a=1%26b=hello%2520plan%25E9te
&_xrd_m=application/atom+xml
```

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

```
https://xri.example.com/=example*r%E9sum%E9/path?query
&_xrd_r=application/xrds+xml;https=true;sep=true
&_xrd_t=http://example.org/test?a=1&b=hello%20plan%E9te
&_xrd_m=application/atom+xml
```

## 11.5 HTTP(S) Accept Headers

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

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

## 11.6 Null Resolution Output Format

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

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

## 11.7 Outputs and HTTP(S) Redirects

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

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

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

This rule enables XRI proxy resolvers to serve clients that do not understand XRI syntax or resolution (such as non-XRI-enabled browsers) by automatically returning a redirect to the service endpoint identified by a combination of the QXRI and the value of the HTTP(S) Accept header (if any) as described in section 0.

## 11.8 Differences Between Proxy Resolution Servers

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

## 11.9 Combining Authority and Proxy Resolution Servers

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

**Comment [DSR5]:** PROOF – This is the only section with substantial updates in ED08. Please proof it carefully.

# 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.

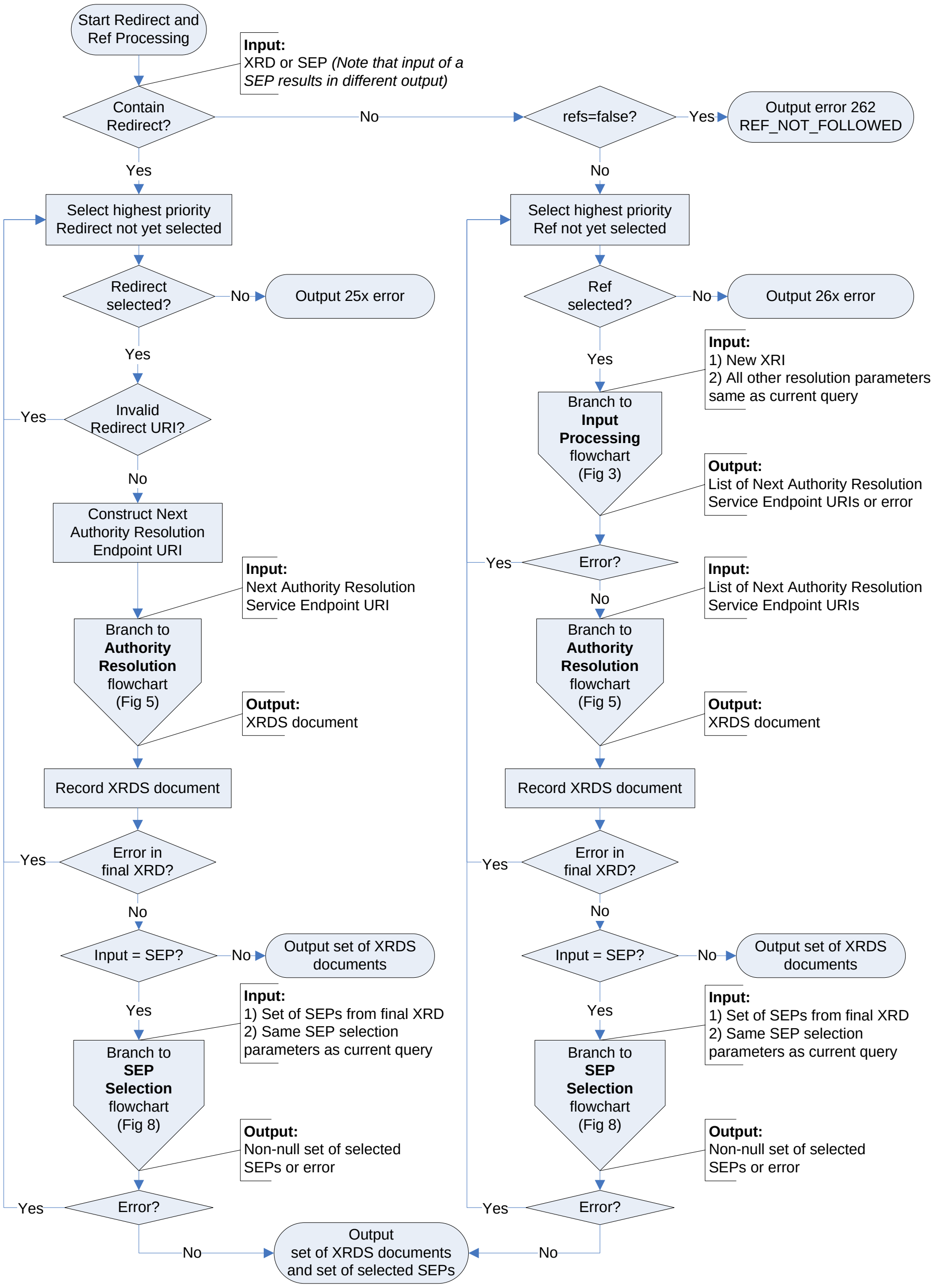
| Requirement | Redirect | Ref |
|---|---|---|
| Must contain | HTTP(S) URI | XRI |
| Accepts the same `append` attribute as the `xrd:URI` element | Yes | No |
| Delegates to a different authority | No – just redirects to a different network location | Yes |
| Must include a subset of the synonyms available in the source XRD | Yes | No |
| Available at both 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 them 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*.

1896 Figure 7 (non-normative) illustrates the logical flow of Redirect and Ref processing.


Start Redirect and Ref Processing
Input:
XRD or SEP (Note that input of a SEP results in different output)
Contain Redirect?
No
refs=false?
Yes
Output error 262 REF_NOT_FOLLOWED
Yes
No
Select highest priority Redirect not yet selected
Select highest priority Ref not yet selected
Redirect selected?
No
Output 25x error
Ref selected?
No
Output 26x error
Yes
Yes
Input:
1) New XRI
2) All other resolution parameters same as current query
Invalid Redirect URI?
Yes
Branch to Input Processing flowchart (Fig 3)
Output:
List of Next Authority Resolution Service Endpoint URIs or error
No
Construct Next Authority Resolution Endpoint URI
Yes
Error?
No
Input:
Next Authority Resolution Service Endpoint URI
Input:
List of Next Authority Resolution Service Endpoint URIs
Branch to Authority Resolution flowchart (Fig 5)
Branch to Authority Resolution flowchart (Fig 5)
Output:
XRDS document
Output:
XRDS document
Record XRDS document
Record XRDS document
Yes
Error in final XRD?
Yes
Error in final XRD?
No
No
Input = SEP?
No
Output set of XRDS documents
Input = SEP?
No
Output set of XRDS documents
Yes
Input:
1) Set of SEPs from final XRD
2) Same SEP selection parameters as current query
Yes
Input:
1) Set of SEPs from final XRD
2) Same SEP selection parameters as current query
Branch to SEP Selection flowchart (Fig 8)
Branch to SEP Selection flowchart (Fig 8)
Output:
Non-null set of selected SEPs or error
Output:
Non-null set of selected SEPs or error
Yes
Error?
Yes
Error?
No
Output set of XRDS documents and set of selected SEPs
No

1897

1898 *Figure 7: Redirect and Ref processing flowchart.*

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

## 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 MUST contain only one of two choices: zero-or-more `xrd:Redirect` or zero-or-more `xrd:Ref` elements.
- At the SEP level, a SEP MUST contain only one of three choices: 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 service endpoint selection 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 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.

Due to 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 and 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 and 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 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 SHOULD be used if the XRD contains other service endpoints or metadata describing the resource. The second option SHOULD only be used 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 in (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 media type parameter `https` is FALSE, or the parameter 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 media type parameter `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 6.3. 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 a) resolution of an `xrd:Redirect` element fails during the authority resolution phase of the original resolution query, OR b) resolution of an `xrd:Redirect` element fails during the optional service endpoint selection phase of the original resolution query OR if 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 orginated 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 media type parameter `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:Redirect` 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 be performed if the value of the `refs` media type parameter (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 immediately return a response with a status code of `262 REF_NOT_FOLLOWED`. 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 a) resolution of an `xrd:Ref` element fails during the authority resolution phase of the original resolution query, OR b) resolution of an `xrd:Ref` element fails during the optional service endpoint selection phase of the original resolution query OR if 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

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

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

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

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

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

### 12.5.1 Redirect Examples

**Example #1:**

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

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

**Example #2:**

In this example the original query identifier is `xri://@a*b*c`. The second XRD contains a Redirect in its authority resolution service endpoint 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. Again, CanonicalIDs are included to illustrate the synonym verification rule.

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

**Example #3:**

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

### 12.5.2 Ref Examples

**Example #1:**

In this example the original query identifier is xri://@a. The first XRD contains an XRD-level Ref to xri://@x*y.

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

**Example #2:**

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

```
<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>
       <Ref>xri://@x*y</Ref>
     </Service>
   </XRD>
   <XRDS ref="xri://@x*y">
```

```
    <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
      <Query>*x</Query>
      <ProviderID>xri://@</ProviderID>
      <CanonicalID>xri://@!7</CanonicalID>      ;XRDS #2 CID #1
      ...
      <Service>
        <Type>xri://$res*auth*($v*2.0)</Type>
        <URI>http://x.example.com/</URI>
      </Service>
    </XRD>
    <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
      <Query>*y</Query>
      <ProviderID>xri://@!7</ProviderID>
      <CanonicalID>xri://@!7!8</CanonicalID>    ;XRDS #2 CID #2
      ...
      <Service>
        <Type>xri://$res*auth*($v*2.0)</Type>
        <URI>http://y.example.com/</URI>
      </Service>
    </XRD>
  </XRDS>
  <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
    <Query>*c</Query>
    <ProviderID>xri://@!1!2</ProviderID>
    <CanonicalID>xri://@!1!2!3</CanonicalID>    ;XRDS #1 CID #3 IS
CHILD OF XRDS #1 CID #2
    ...
    <Service>
    ...final service endpoints described here...
    </Service>
  </XRD>
</XRDS>
```

**Example #3:**

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

```
<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*($v*2.0)</Type>
      <URI>http://a.example.com/</URI>
    </Service>
  </XRD>
  <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
    <Query>*c</Query>
    <ProviderID>xri://@!1!2</ProviderID>
    <CanonicalID>xri://@!1!2!3</CanonicalID>      ;XRDS #1 CID #3
    ...
    <Service>
      <Type>http://openid.net/signon/1.0</Type>
      <Ref>xri://@x*y</Ref>
    </Service>
  </XRD>
  <XRDS ref="xri://@x*y">
    <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
      <Query>*x</Query>
      <ProviderID>xri://@</ProviderID>
      <CanonicalID>xri://@!7</CanonicalID>        ;XRDS #2 CID #1
      ...
      <Service>
        <Type>xri://$res*auth*($v*2.0)</Type>
        <URI>http://x.example.com/</URI>
      </Service>
    </XRD>
    <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
      <Query>*y</Query>
      <ProviderID>xri://@!7</ProviderID>
      <CanonicalID>xri://@!7!8</CanonicalID>      ;XRDS #2 CID #2
      ...
      <Service>
        <Type>xri://$res*auth*($v*2.0)</Type>
        <URI>http://y.example.com/</URI>
      </Service>
      <Service>
        <Type>http://openid.net/signon/1.0</Type>
        <URI>http://openid.example.com/</URI>
      </Service>
    </XRD>
  </XRDS>
</XRDS>
```

## 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 is separate and distinct from Redirect and Ref processing invoked during the optional service endpoint selection phase. 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 requeste service endpoint or else return an error, i.e., it never backtracks into the authority resolution phase.

- *First recursion point.* The first time a resolver first encounters a Redirect or a Ref within a phase is called the *first recursion point*. There can 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, the resolver is obligated to resolve the highest priority Redirect or Ref to see if it can satisfy the resolution query. If the first Redirect or Ref fails during the authority resolution phase, the resolver MUST continue trying the next highest priority Redirect or Ref 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. If the first Redirect or Ref fails during the optional service endpoint selection phase, the resolver MUST continue trying the next highest priority Redirect or Ref until either it 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 any next recursion point completely fails, 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.

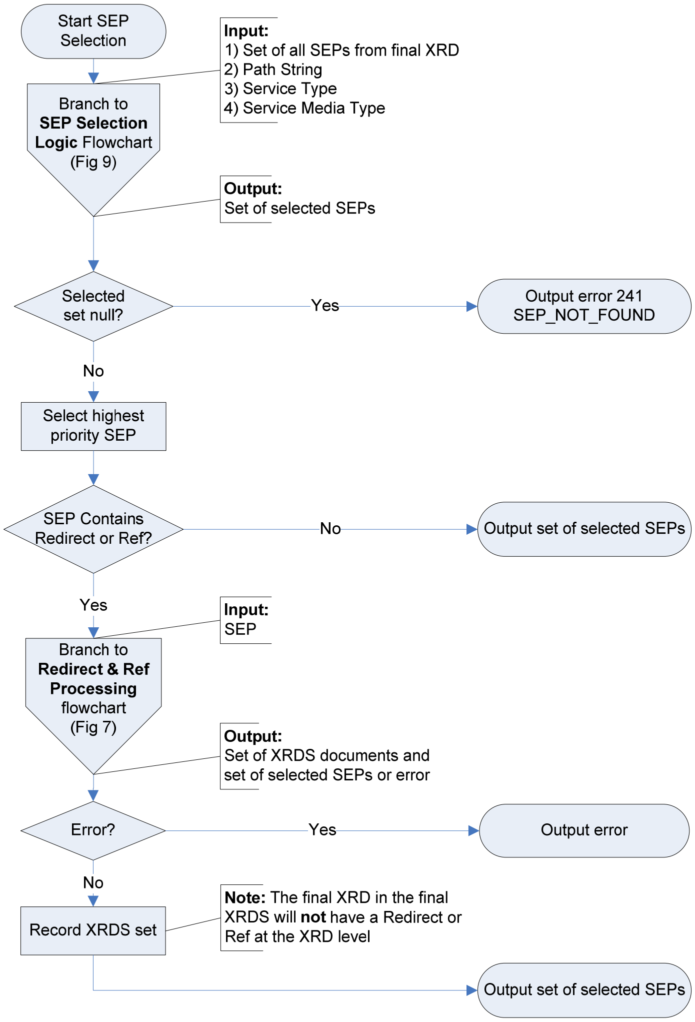
To avoid excessive recursion and inefficient resolution responses, XRDS authors are RECOMMENDED to use as few Redirects or Refs in a resolution chain as possible.

# 13 Service Endpoint Selection

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

## 13.1 Processing Rules

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 set of selected SEPs
Yes
Input:
SEP
Branch to Redirect & Ref Processing flowchart (Fig 7)
Output:
Set of XRDS documents and set of selected SEPs or error
Error?
Yes
Output error
No
Record XRDS set
Note: The final XRD in the final XRDS will not have a Redirect or Ref at the XRD level
Output set of selected SEPs

*Figure 8: Service endpoint selection flowchart.*

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

1. The inputs for service endpoint selection are defined in Table 8.
2. If the final XRD resulting from authority resolution contains an `xrd:XRD/xrd:Redirect` or `xrd:XRD/xrd:Ref` element, it MUST first be processed as specified in section 12.
3. The set of all service endpoints (`xrd:XRD/xrd:Service` elements) in the final XRD, selection MUST be processed according to the service endpoint selection logic defined in section 13.2. The output of this process will be either the null set or a selected set of one or more service endpoints.
4. If, after applying the service endpoint selection logic, the selected set is not null and the highest priority selected service endpoint contains an `xrd:XRD/xrd:Service/xrd:Redirect` or `xrd:XRD/xrd:Service/xrd:Ref` element, it MUST first be processed as specified in section 12. This is a recursive call that will produce a nested XRDS document.
5. If, after applying the service endpoint selection logic, the selected set is null, the resolver MUST return the error `221 AUTH_RES_NOT_FOUND` if authority resolution is not complete, or `221 SEP_NOT_FOUND` if authority resolution is complete.
6. If authority resolution is complete, the output of service endpoint selection MUST be returned in a valid Resolution Output Format as defined in Table 10 and conform to the output requirements defined in section 8.2.

## 13.2 Service Endpoint Selection Logic

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

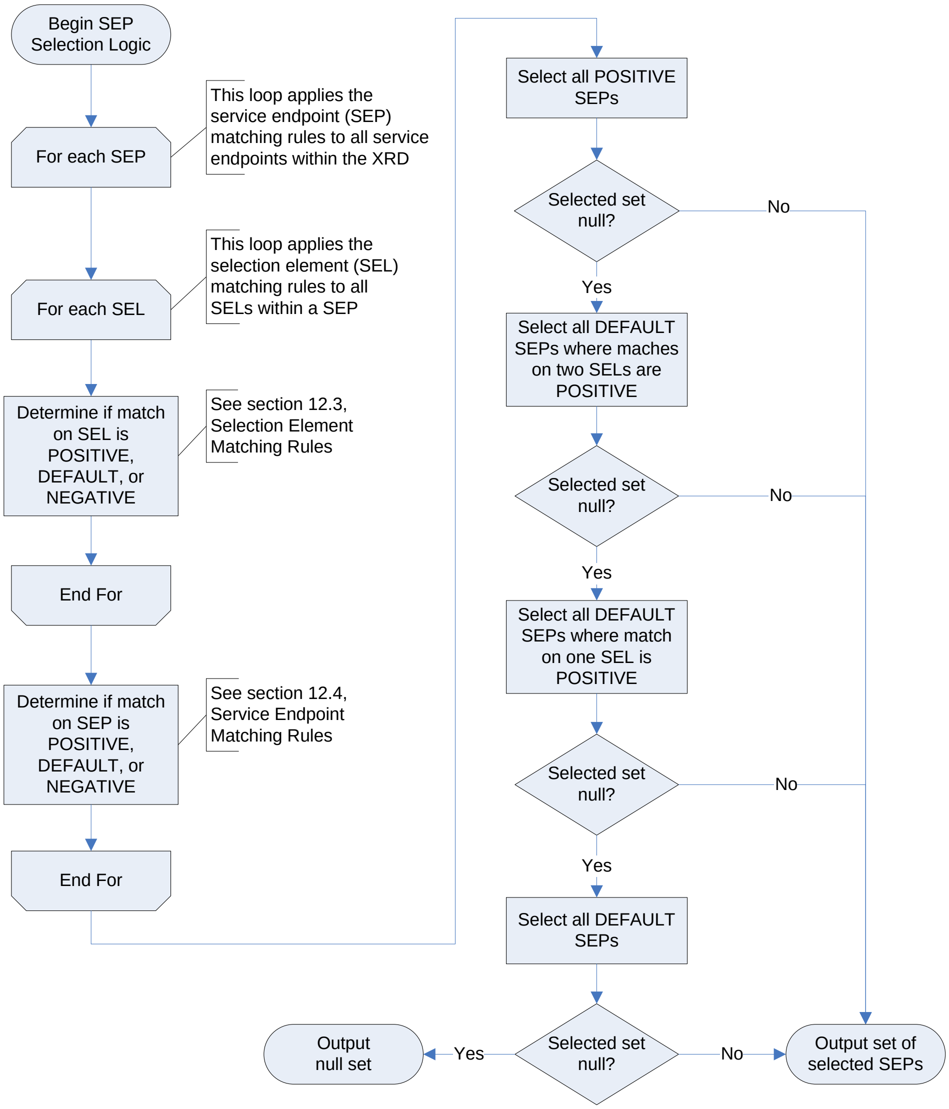

Begin SEP Selection Logic
For each SEP
This loop applies the service endpoint (SEP) matching rules to all service endpoints within the XRD
For each SEL
This loop applies the selection element (SEL) matching rules to all SELs within a SEP
Determine if match on SEL is POSITIVE, DEFAULT, or NEGATIVE
See section 12.3, Selection Element Matching Rules
End For
Determine if match on SEP is POSITIVE, DEFAULT, or NEGATIVE
See section 12.4, Service Endpoint Matching Rules
End For
Select all POSITIVE SEPs
Selected set null?
No
Yes
Select all DEFAULT SEPs where maches on two SELs are POSITIVE
Selected set null?
No
Yes
Select all DEFAULT SEPs where match on one SEL is POSITIVE
Selected set null?
No
Yes
Select all DEFAULT SEPs
Selected set null?
Yes
Output null set
No
Output set of selected SEPs

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

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

## 13.3 Selection Element Matching Rules

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

### 13.3.1 Selection Element Match Options

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

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

*Table 24: Match options for selection elements.*

The Precedence order is used in the Multiple Selection Element Matching Rule (section 13.3.5). It is important to note that failure of a POSITIVE match does not necessarily mean a NEGATIVE match; it may still qualify as a DEFAULT match.

### 13.3.2 The Match Attribute

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

| Value | Matching Rule Applied to Corresponding Input Parameter |
|---|---|
| any | Automatically a POSITIVE match (i.e., input parameter is ignored). |
| default | Automatically a DEFAULT match (i.e., input parameter is ignored) UNLESS the value of the Resolution Output Format nodefault_t, nodefault_p or nodefault_m parameter 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. |

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

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

### 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 the 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. 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 prior to input. 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. Also, if the value is an XRI or IRI it MUST be in URI-normal form as defined in section 4.4. As a best practice, service architects SHOULD assign identifiers for service types that are easy to match, are in URI-normal form, and do not require further normalization.
2. 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.

### 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. The Path String being matched MUST include the leading forward slash separating an XRI authority segment from the path. 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 segment from the path. If it does not, one MUST be prepended before 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.

Examples of this rule are shown in Table 26.

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

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

### 13.3.8 MediaType Element Matching Rules

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

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

## 13.4 Service Endpoint Matching Rules

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

### 13.4.1 Service Endpoint Match Options

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. |

*Table 27: Match options for service endpoints.*

### 13.4.2 Select Attribute Match Rule

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

### 13.4.3 All Positive Match Rule

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

### 13.4.4 Default Match Rule

If a service endpoint fails the Select Attribute Match Rule and the All Positive Match Rule, but none of the three categories of selection elements has a NEGATIVE match as defined in section 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.Type
- Postive.Path
- Positive.MediaType
- Default.Type
- Default.Path
- Default.MediaType
- Present.Type
- Present.Path
- Present.MediaType

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.

**Comment [DSR6]:** PROOF – This is the new compact pseudocode crafted by Wil – it fits on one page!

```
FOR EACH SEP
   CREATE set of SEL match flags
   SET all flags to FALSE
   FOR EACH SEL of category x (where x=Type, Path, or Mediatype)
         SET Present.x=TRUE
         IF match on this SEL is POSITIVE
               IF select="true"                  ;see 12.4.2
                     ADD SEP TO SELECTED SET
                     NEXT SEP
               ELSE
                     SET Positive.x=TRUE
               ENDIF
         ELSEIF match on this SEL is DEFAULT     ;see 10.3.2 & 12.3.4
               IF Positive.x != TRUE AND
               nodefault != x                    ;see 12.3.5
                     SET Default.x=TRUE
               ENDIF
         ENDIF
   ENDFOR
   IF Present.x=FALSE                            ;see 12.3.3
         IF nodefault_x != TRUE                  ;see 10.3.2
               SET Default.x=TRUE
         ENDIF
   ENDIF
   IF Positive.Type=TRUE AND
      Positive.Path=TRUE AND
      Positive.Mediatype=TRUE                    ;see 12.4.3
         ADD SEP TO SELECTED SET
         NEXT SEP
   ELSEIF SELECTED SET != EMPTY                  ;see 12.5.1
         NEXT SEP
   ELSEIF (Positive.Type=TRUE OR Default.Type=TRUE) AND
          (Positive.Path=TRUE OR Default.Path=TRUE) AND
          (Positive.MediaType=TRUE OR Default.MediaType=TRUE)
         ADD SEP TO DEFAULT SET                  ;see 12.4.4
   ENDIF
ENDFOR
IF SELECTED SET = EMPTY                          ;see 12.5.1
   FOR EACH SEP IN DEFAULT SET
         IF (Positive.Type=TRUE AND Positive.Path=TRUE) OR
         (Positive.Type=TRUE AND Positive.MediaType=TRUE) OR
         (Positive.Path=TRUE AND Positive.MediaType=TRUE)
               ADD SEP TO SELECTED SET
         ENDIF
   ENDFOR
   IF SELECTED SET = EMPTY
         FOR EACH SEP IN DEFAULT SET
               IF Positive.Type=TRUE OR
               Positive.Path=TRUE OR
               Positive.MediaType=TRUE
                     ADD SEP TO SELECTED SET
               ENDIF
         ENDFOR
   ENDIF
ENDIF
IF SELECTED SET != EMPTY
   RETURN SELECTED SET
ELSE
   RETURN DEFAULT SET
ENDIF
```

## 13.7 Construction of Service Endpoint URIs

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

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

### 13.7.1 The append Attribute

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

| Value | Component of QXRI to Append |
|---|---|
| none | None. This is the default if the `append` attribute is absent |
| local | The entire local part of the QXRI, defined as being one of three cases:<br>a) If only a path is present, the path string *including the leading forward slash*<br>b) If only a query is present, the query string *including the leading question mark*<br>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*<br>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 |

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

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

1. If the value is `none`, the exact contents of the `xrd:URI` element MUST be returned directly without any further processing.
2. For any other value, the exact value in URI-normal form of the QXRI component specified in Table 28, *including any leading delimiter(s)* and *without any additional escaping or percent encoding* MUST be appended directly to the exact contents of the `xrd:URI` element *including any trailing delimiter(s)*. If the value of the QXRI component specified in Table 28 consists of only a leading delimiter, then this value MUST be appended according to these rules. If the value of the QXRI component specified in Table 28 is null, then the contents of the `xrd:URI` element MUST be returned directly exactly as if the value of the `append` attribute was `none`.
3. If any HXRI query parameters for proxy resolution were added to an existing QXRI query component as defined in section 11.3, these query parameters MUST be removed prior

to performing the append operation as also defined in section 11.3. In particular, if after removal of these query parameters the QXRI query component consists of only *a string of one or more question marks* (the delimiting question mark plus zero or more additional question marks) then *exactly one question mark* MUST also be removed. This preserves the query component of the original QXRI if it was null or contained only question marks.

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

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

### 13.7.2 The uric Parameter

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

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

# 14 Synonym Verification

As described in section 5, *XRD Synonym Elements,* 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 a set of synonym verification rules.

## 14.1 Redirect Verification

**Comment [DSR7]:** PROOF – This section was moved here from section 12.3 because it fit here better and could be referenced from the flowcharts

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>
```

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

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

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

## 14.3 CanonicalID Verification

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

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

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

### 14.3.1 HTTP(S) URI Verification Rules

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

1. The query identifier MUST 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 query identifier, or b) the query identifier plus a valid fragment as defined by **[RFC3986]**.

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

### 14.3.2 XRI Verification Rules

To verify that an XRI is a valid CanonicalID synonym for a query identifier, an XRI 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:ProviderID` element in the XRD from the community root authority MUST match the value of the `xrd:XRD/xrd:CanonicalID` element configured in the XRI resolver or available in a self-describing XRD from the community root authority (or its equivalent). See section 9.1.6.
2. In the first XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID` element MUST be a direct child authority of the value of the `xrd:XRD/xrd:ProviderID` element. i.e., the former MUST consist of the latter plus one additional XRI subsegment as defined in **[XRISyntax]**. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element is `@!1`, then the the value of the `xrd:XRD/xrd:ProviderID` element must be `@`.
3. For each subsequent XRD in the resolution chain, the value of the `xrd:XRD/xrd:CanonicalID` element MUST be a direct child authority of the value of the `xrd:XRD/xrd:CanonicalID` element in the parent XRD. For example, if the value of the `xrd:XRD/xrd:CanonicalID` element asserted in an XRD is `@!1!2!3`, then the value of the `xrd:XRD/xrd:CanonicalID` element in the XRD of the parent authority must be `@!1!2`.
4. If Redirect or Ref processing is required during resolution as specified in section 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

**Comment [DSR8]:** PROOF – The changes in this section fixed a bug: resolution of a CanonicalEquivID can, due to Redirect or Ref processing, return the same XRD asserting the CanonicalEquivID, and that form of verification is as valid as obtaining an EquivID backpointer.

CanonicalID verification also requires verification of a CanonicalEquivID *only if it is present in the final XRD in the resolution chain*. Since CanonicalEquivID verification 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 is accomplished by resolving the CanonicalEquivID and verifying that it either resolves to the XRD asserting the CanonicalEquivID or to an XRD asserting an EquivID backpointer to the XRD asserting the CanonicalEquivID. To verify that either an HTTP(S) URI or an XRI is a valid hierachical CanonicalEquivID synonym for a query identifier, an XRI resolver MUST verify that all the following tests are successful:

**Deleted:** that

**Deleted:** exists in the XRD obtained by resolving the CanonicalEquivID value.

1. CanonicalID verification as specified in section 14.3 MUST have completed successfully.
2. The asserted CanonicalEquivID value MUST be a valid HTTP(S) URI or XRI.
3. The asserted CanonicalEquivID value MUST resolve successfully to an XRDS document according to the rules in this specification *using the same resolution parameters as in the original resolution request*.

4. The final XRD in the XRDS document MUST either: a) be the same XRD asserting the CanonicalEquivID synonym, or b) contain a `xrd:XRD/xrd:EquivID` element whose value is equivalent to the value of the verified `xrd:XRD/xrd:CanonicalID` element in the XRD asserting the CanonicalEquivID synonym.

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

### 14.3.4 Verification Status Attributes

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

1. CanonicalID verification MUST be reported using the `cid` attribute.
2. CanonicalEquivID verification MUST be reported using the `ceid` attribute.
3. Both attributes accept four enumerated values: `absent` if the element is not present, `off` if verification is not performed, `verified` if the element is verified, and `failed` if verification fails.
4. The `off` value applies to both elements if CanonicalID verification is not performed (`cid=false`).
5. The `off` value applies to the CanonicalEquivID element in any XRD before the final XRD if CanonicalID verification is performed (`cid=true`), because a resolver only verifies this element in the final XRD.

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

### 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>
  <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>
  <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.

---

**Example #3:**

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

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

```
<XRDS>
  <XRD>
    <CanonicalID>http://example.com/user</CanonicalID>
    <CanonicalEquivID>
     https://different.example.net/path/user
   </CanonicalEquivID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

```
<XRDS>
  <XRD>
    <EquivID>http://example.com/user</EquivID>
    <CanonicalID>https://different.example.net/path/user</CanonicalID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

---

**Example #4:**

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

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

```
<XRDS>
  <XRD>
    <CanonicalID>http://example.com/user</CanonicalID>
    <CanonicalEquivID>xri://=!1000.62b1.44fd.2855</CanonicalEquivID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

```
<XRDS>
  <XRD>
    <Query>!1000.62b1.44fd.2855</Query>
    <ProviderID>xri://=</ProviderID>
    <EquivID>http://example.com/user</EquivID>
    <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

---

**Example #5:**

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

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

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

```
    <Query>*example.name</Query>
    <ProviderID>xri://=</ProviderID>
    <LocalID>!1000.62b1.44fd.2855</LocalID>
    <CanonicalID>xri://=!1000.62b1.44fd.2855</CanonicalID>
    <CanonicalEquivID>https://example.com/user</CanonicalEquivID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

```
<XRDS>
  <XRD>
    <EquivID>xri://=!1000.62b1.44fd.2855</EquivID>
    <CanonicalID>https://example.com/user</CanonicalID>
    <Service priority="10">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

---

**Example #6:**

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

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

```
<XRDS>
  <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>
    <CanonicalEquivID>
     xri://@11000.f3da.9056.aca3!5555
    </CanonicalEquivID>
    <Service priority="1">
       ...
    </Service>
    ...
  </XRD>
</XRDS>
```

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

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

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

# 15 Status Codes and Error Processing

## 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. (See the backwards compatability note below.)
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.

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

## 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.

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

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

- x1x: Input error
- x2x: Generic authority resolution error.
- x3x: Trusted authority resolution error.
- x4x: Service endpoint (SEP) selection error.
- x5x: Redirect error.
- x6x: Ref error.

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

| Code | Symbolic Status | Phase(s) | Description |
|---|---|---|---|
| 100 | SUCCESS | Any | Operation was successful. |
| 200 | PERM_FAIL | Any | Generic permanent failure. |
| 201 | NOT_IMPLEMENTED | Any | The requested function (trusted resolution, service endpoint selection) is not implement by the resolver. |
| 202 | LIMIT_EXCEEDED | Any | A locally configured resource limit was exceeded. Examples: number of references to follow, number of XRD elements that can be handled, size of an XRDS document. |
| 210 | INVALID_INPUT | Input | Generic input error. |
| 211 | INVALID_QXRI | Input | Input QXRI does not conform to XRI syntax. |
| 212 | INVALID_OUTPUT_FORMAT | Input | Input Resolution Output Format is invalid. |
| 213 | INVALID_SEP_TYPE | Input | Input Service Type is invalid. |
| 214 | INVALID_SEP_MEDIA_TYPE | Input | Input Service Media Type is invalid. |
| 215 | UNKNOWN_ROOT | Input | Community root specified in QXRI is not configured in the resolver. |
| 220 | AUTH_RES_ERROR | Authority resolution | Generic authority resolution error. |
| 221 | AUTH_RES_NOT_FOUND | Authority resolution | The subsegment cannot be resolved due to a missing authority resolution service endpoint in an XRD. |
| 222 | QUERY_NOT_FOUND | Authority resolution | Responding authority does not have an XRI matching the query. |
| 223 | UNEXPECTED_XRD | Authority resolution | Value of the `xrd:Query` element does not match the subsegment requested. |
| 224 | INACTIVE | Authority resolution | The query XRI has been assigned but the authority does not provide resolution metadata. |
| 230 | TRUSTED_RES_ERROR | Trusted | Generic trusted resolution error. |

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

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

*Table 29: Error codes for XRI resolution.*

## 15.3 Status Context Strings

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 the second line of a plain text message as specified in section 11.7. 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 OPTIONAL; 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 understanding 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 service endpoint selection, 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 segment 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 a 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>
      <ProviderID>…</ProviderID>
      …
      <other:ExtensionElement>…</other:ExtensionElement>
    </Service>
  </other:SuperService>
</XRD>
```

**Comment [DSR9]:** PROOF – Gabe, wasn’t this element missing? See the text that follows this example.

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” so only consumers that understand elements in the `other:SuperService` namespace will attempt to process the `xrd:ProviderID` element.

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

### 17.1.2 Other Points of Extensibility

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

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

## 17.2 Versioning

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

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

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

### 17.2.1 Version Numbering

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

$\mathrm{Major}_B > \mathrm{Major}_A$ OR ( ( $\mathrm{Major}_B = \mathrm{Major}_A$ ) AND $\mathrm{Minor}_B > \mathrm{Minor}_A$ )

### 17.2.2 Versioning of the XRI Resolution Specification

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

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

### 17.2.3 Versioning of Protocols

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

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

### 17.2.4 Versioning of XRDs

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

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

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

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

# 18 Security and Data Protection

Significant portions of this specification deal directly with security issues, and these will not be summarized again here. In addition, basic security practices and typical risks in resolution protocols are well-documented in many other specifications. Only security considerations directly relevant to XRI resolution are included here.

## 18.1 DNS Spoofing or Poisoning

When XRI resolution is deployed to use HTTP URIs or other URIs which include DNS names, the accuracy of the XRI resolution response may be dependent on the accuracy of DNS queries. For those deployments where DNS is not trusted, the resolution infrastructure may be deployed with HTTP URIs that use IP addresses in the authority portion of HTTP URIs and/or with the trusted resolution mechanisms defined by this specification. Resolution results obtained using trusted resolution can be evaluated independently of DNS resolution results. While this does not solve the problem of DNS spoofing, it does allow the client to detect an error condition and reject the resolution result as untrustworthy. In addition, **[DNSSEC]** may be considered if DNS names are used in HTTP URIs.

## 18.2 HTTP Security

Many of the security considerations set forth in HTTP/1.1 **[RFC2616]** apply to XRI Resolution protocols defined here.  In particular, confidentiality of the communication channel is not guaranteed by HTTP. Server-authenticated HTTPS should be used in cases where confidentiality of resolution requests and responses is desired.

Special consideration should be given to proxy and caching behaviors to ensure accurate and reliable responses from resolution requests. For various reasons, network topologies increasingly have transparent proxies, some of which may insert VIA and other headers as a consequence, or may even cache content without regard to caching policies set by a resource’s HTTP authority.

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

## 18.3 SAML Considerations

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

## 18.4 Limitations of Trusted Resolution

While the trusted resolution protocols specified in this document provides a way to verify the integrity of a successful XRI resolution, it may not provide a way to verify the integrity of a resolution failure. Reasons for this limitation include the prevalence of non-malicious network failures, the existence of denial-of-service attacks, and the ability of a man-in-the-middle attacker to modify HTTP responses when resolution is not performed over HTTPS.

Additionally, there is no revocation mechanism for the keys used in trusted resolution. Therefore, a signed resolution's validity period should be limited appropriately to mitigate the risk of an incorrect or invalid resolution.

## 18.5 Synonym Verification

As discussed in section 5, XRI and XRDS infrastructure has rich support for identifiers synonyms, including identifier synonyms that cross security domains. For this reason it is particularly important that identifier authorities, including registries, registrars, directory administrators, identity brokers, 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 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 segment 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 the current 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.

# B. RelaxNG Schema for XRDS and XRD

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

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

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

### xrds.rnc

```
namespace xrds = "xri://$xrds"
namespace xrd = "xri://$xrd*($v*2.0)"
namespace local = ""
datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"

any.element =
  element *  {
    (attribute * { text } *
     | text
     | any.element)*
  }

any.external.element =
  element * - (xrd:XRD | xrds:XRDS)  {
    (attribute * { text } *
     | text
     | any.element)*
  }

other.attribute = attribute * - (local:*) {text}

start = XRDS

XRDS = element xrds:XRDS {
    other.attribute *,
    (attribute ref { xs:anyURI } | attribute redirect { xs:anyURI} )?,
    (any.external.element  | XRDS | external "xrd.rnc" )*
}
```

### xrd.rnc

```
default namespace = "xri://$xrd*($v*2.0)"
namespace xrd = "xri://$xrd*($v*2.0)"
namespace saml = "urn:oasis:names:tc:SAML:2.0:assertion"
namespace ds = "http://www.w3.org/2000/09/xmldsig#"
namespace local = ""

datatypes xs = "http://www.w3.org/2001/XMLSchema-datatypes"

start = XRD

anyelementbody =
    (attribute * {text}
    | text
    | element * {anyelementbody} )*

non.xrd.element = element * - xrd:* {
    anyelementbody
}

other.attribute = attribute * - (local:* | xrd:* ) {text}


XRD = element XRD {
    other.attribute *,
    attribute idref {xs:IDREF} ?,
    attribute version { "2.0" } ?,
    Query ?,
    Status ?,
    ServerStatus ?,
    Expires ?,
    ProviderID ?,
    (Redirect | Ref) ?,
    LocalID *,
    EquivID *,
    CanonicalID ?,
    CanonicalEquivID ?,
    Service *,
    element saml:Assertion {anyelementbody} ?,
    non.xrd.element *
}

Query = element Query {
    other.attribute *,
    text
}

statuspattern =
    other.attribute *,
    attribute code {xs:integer},
    attribute cid { "absent" | "off" | "verified" | "failed" } ?,
    attribute ceid { "absent" | "off" | "verified" | "failed" } ?,
    text

Status = element Status {
    statuspattern
}

ServerStatus = element ServerStatus {
    statuspattern
}

Expires = element Expires {
    other.attribute *,
    xs:dateTime
}

ProviderID = element ProviderID {
    other.attribute *,
    xs:anyURI
}

Redirect = element Redirect {
    other.attribute *,
    attribute priority {xs:integer}?,
    xs:anyURI
}

Ref = element Ref{
    other.attribute *,
    attribute priority {xs:integer}?,
    xs:anyURI
}

LocalID = element LocalID {
    other.attribute *,
    attribute priority {xs:integer} ?,
    xs:anyURI
}

EquivID = element EquivID {
    other.attribute *,
    attribute priority {xs:integer} ?,
    xs:anyURI
}
```

```

CanonicalID = element CanonicalID {
    other.attribute *,
    xs:anyURI
}

CanonicalEquivID = element CanonicalEquivID {
    other.attribute *,
    xs:anyURI
}

Service = element Service {
    other.attribute *,
    attribute priority {xs:integer}?,
    ProviderID?,
    Type *,
    Path *,
    MediaType *,
    (URI+|Redirect+|Ref+)?,
    LocalID *,
    element ds:KeyInfo {anyelementbody}?,
    non.xrd.element *
}

URI = element URI {
    other.attribute *,
    attribute priority {xs:integer}?,
    attribute append {"none" | "local" | "authority" | "path" | "query" |

"qxri"} ?,
    xs:anyURI
}

selection.attributes = attribute match {"any" | "default" | "non-null" |

"null" } ?,
                        attribute select { xs:boolean} ?

Type = element Type {
    other.attribute *,
    selection.attributes,
    xs:anyURI
}

Path = element Path {
    other.attribute *,
    selection.attributes,
    xs:string
}

MediaType = element MediaType {
    other.attribute *,
    selection.attributes,
    xs:string
}
```

# C. XML Schema for XRDS and XRD

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

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

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

XRDS.XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds"
targetNamespace="xri://$xrds" elementFormDefault="qualified">
        <!-- Utility patterns -->
        <xs:attributeGroup name="otherattribute">
                <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:attributeGroup>
        <xs:group name="otherelement">
                <xs:choice>
                        <xs:any namespace="##other" processContents="lax"/>
                        <xs:any namespace="##local" processContents="lax"/>
                </xs:choice>
        </xs:group>
        <!-- Patterns for elements -->
        <xs:element name="XRDS">
                <xs:complexType>
                        <xs:sequence>
                                <xs:group ref="xrds:otherelement" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attributeGroup ref="xrds:otherattribute"/>
                        <!--XML Schema does not currently offer a means to express that
only one of the following two attributes may be used in any XRDS element, i.e., an XRDS
document may describe EITHER a redirect identifier or a ref identifier but not both.-->
                        <xs:attribute name="redirect" type="xs:anyURI" use="optional"/>
                        <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

XRD.XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xrd="xri://$xrd*($v*2.0)"
targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified">
        <!-- Utility patterns -->
        <xs:attributeGroup name="otherattribute">
                <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:attributeGroup>
        <xs:group name="otherelement">
                <xs:choice>
                        <xs:any namespace="##other" processContents="lax"/>
                        <xs:any namespace="##local" processContents="lax"/>
                </xs:choice>
        </xs:group>
        <xs:attributeGroup name="priorityAttrGrp">
                <xs:attribute name="priority" type="xs:nonNegativeInteger"
use="optional"/>
        </xs:attributeGroup>
        <xs:attributeGroup name="codeAttrGrp">
```

```
                <xs:attribute name="code" type="xs:int" use="required"/>
        </xs:attributeGroup>
        <xs:attributeGroup name="verifyAttrGrp">
                <xs:attribute name="cid" use="optional">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:enumeration value="absent"/>
                                        <xs:enumeration value="off"/>
                                        <xs:enumeration value="verified"/>
                                        <xs:enumeration value="failed"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="ceid" use="optional">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:enumeration value="absent"/>
                                        <xs:enumeration value="off"/>
                                        <xs:enumeration value="verified"/>
                                        <xs:enumeration value="failed"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
        </xs:attributeGroup>
        <xs:attributeGroup name="selectionAttrGrp">
                <xs:attribute name="match" use="optional" default="default">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:enumeration value="default"/>
                                        <xs:enumeration value="any"/>
                                        <xs:enumeration value="non-null"/>
                                        <xs:enumeration value="null"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="select" type="xs:boolean" use="optional"
default="false"/>
        </xs:attributeGroup>
        <xs:attributeGroup name="appendAttrGrp">
                <xs:attribute name="append" use="optional" default="none">
                        <xs:simpleType>
                                <xs:restriction base="xs:string">
                                        <xs:enumeration value="none"/>
                                        <xs:enumeration value="local"/>
                                        <xs:enumeration value="authority"/>
                                        <xs:enumeration value="path"/>
                                        <xs:enumeration value="query"/>
                                        <xs:enumeration value="qxri"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
        </xs:attributeGroup>
        <xs:complexType name="URIPattern">
                <xs:simpleContent>
                        <xs:extension base="xs:anyURI">
                                <xs:attributeGroup ref="xrd:otherattribute"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
        <xs:complexType name="URIPriorityPattern">
                <xs:simpleContent>
                        <xs:extension base="xrd:URIPattern">
                                <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
        <xs:complexType name="URIPriorityAppendPattern">
                <xs:simpleContent>
                        <xs:extension base="xrd:URIPriorityPattern">
                                <xs:attributeGroup ref="xrd:appendAttrGrp"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
        <xs:complexType name="StringPattern">
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attributeGroup ref="xrd:otherattribute"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
        <xs:complexType name="StringSelectionPattern">
                <xs:simpleContent>
                        <xs:extension base="xrd:StringPattern">
                                <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
        <!-- Patterns for elements -->
        <xs:element name="XRD">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="xrd:Query" minOccurs="0"/>
                                <xs:element ref="xrd:Status" minOccurs="0"/>
                                <xs:element ref="xrd:ServerStatus" minOccurs="0"/>
                                <xs:element ref="xrd:Expires" minOccurs="0"/>
                                <xs:element ref="xrd:ProviderID" minOccurs="0"/>
                                <xs:choice>
                                        <xs:element ref="xrd:Redirect" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element ref="xrd:Ref" minOccurs="0"
maxOccurs="unbounded"/>
                                </xs:choice>
                                <xs:element ref="xrd:LocalID" minOccurs="0"
maxOccurs="unbounded"/>
                                <xs:element ref="xrd:EquivID" minOccurs="0"
maxOccurs="unbounded"/>
                                <xs:element ref="xrd:CanonicalID" minOccurs="0"
maxOccurs="unbounded"/>
                                <xs:element ref="xrd:CanonicalEquivID" minOccurs="0"
maxOccurs="unbounded"/>
                                <xs:element ref="xrd:Service" minOccurs="0"
maxOccurs="unbounded"/>
                                <xs:group ref="xrd:otherelement" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:ID"/>
                        <xs:attribute name="idref" type="xs:IDREF" use="optional"/>
                        <xs:attribute name="version" type="xs:string" use="optional"
fixed="2.0"/>
                        <xs:attributeGroup ref="xrd:otherattribute"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="Query" type="xrd:StringPattern"/>
        <xs:element name="Status">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xrd:StringPattern">
                                        <xs:attributeGroup ref="xrd:codeAttrGrp"/>
                                        <xs:attributeGroup ref="xrd:verifyAttrGrp"/>
                                        <xs:attributeGroup ref="xrd:otherattribute"/>
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
        <xs:element name="ServerStatus">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xrd:StringPattern">
                                        <xs:attributeGroup ref="xrd:codeAttrGrp"/>
                                        <xs:attributeGroup ref="xrd:otherattribute"/>
                                </xs:extension>
                        </xs:simpleContent>
```

**Comment [DSR10]:** OPEN ISSUE – see section D2 of http://www.w3.org/TR/xml-id/.

```
                        </xs:complexType>
                </xs:element>
                <xs:element name="Expires">
                        <xs:complexType>
                                <xs:simpleContent>
                                        <xs:extension base="xs:dateTime">
                                                <xs:attributeGroup ref="xrd:otherattribute"/>
                                        </xs:extension>
                                </xs:simpleContent>
                        </xs:complexType>
                </xs:element>
                <xs:element name="ProviderID" type="xrd:URIPattern"/>
                <xs:element name="LocalID">
                        <xs:complexType>
                                <xs:simpleContent>
                                        <xs:extension base="xrd:StringPattern">
                                                <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
                                        </xs:extension>
                                </xs:simpleContent>
                        </xs:complexType>
                </xs:element>
                <xs:element name="EquivID" type="xrd:URIPriorityPattern"/>
                <xs:element name="CanonicalID" type="xrd:URIPriorityPattern"/>
                <xs:element name="CanonicalEquivID" type="xrd:URIPriorityPattern"/>
                <xs:element name="Redirect" type="xrd:URIPriorityAppendPattern"/>
                <xs:element name="Ref" type="xrd:URIPriorityPattern"/>
                <xs:element name="Service">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:element ref="xrd:ProviderID" minOccurs="0"/>
                                        <xs:element ref="xrd:Type" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element ref="xrd:Path" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:element ref="xrd:MediaType" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:choice>
                                                <xs:element ref="xrd:URI" minOccurs="0"
maxOccurs="unbounded"/>
                                                <xs:element ref="xrd:Redirect" minOccurs="0"
maxOccurs="unbounded"/>
                                                <xs:element ref="xrd:Ref" minOccurs="0"
maxOccurs="unbounded"/>
                                        </xs:choice>
                                        <xs:element ref="xrd:LocalID" minOccurs="0"
maxOccurs="unbounded"/>
                                        <xs:group ref="xrd:otherelement" minOccurs="0"
maxOccurs="unbounded"/>
                                </xs:sequence>
                                <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
                                <xs:attributeGroup ref="xrd:otherattribute"/>
                        </xs:complexType>
                </xs:element>
                <xs:element name="Type">
                        <xs:complexType>
                                <xs:simpleContent>
                                        <xs:extension base="xrd:URIPattern">
                                                <xs:attributeGroup ref="xrd:selectionAttrGrp"/>
                                        </xs:extension>
                                </xs:simpleContent>
                        </xs:complexType>
                </xs:element>
                <xs:element name="MediaType" type="xrd:StringSelectionPattern"/>
                <xs:element name="Path" type="xrd:StringSelectionPattern"/>
                <xs:element name="URI" type="xrd:URIPriorityAppendPattern"/>
</xs:schema>
```

# 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

# F. Example Local Resolver Interface Definition (Informative)

**Comment [DSR11]:** PROOF – This has been updated by Wil Tan and Steve Churchill

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

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

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

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

| Config Parameter | Description |
|---|---|
| `https,`<br>`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,`<br>`nodefault_p,`<br>`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. |

Note that one subparameter defined in in Table 6, `sep` (service endpoint), is not included in this flags table because it is implicitly represented in the operation being called. The five operations shown in the table below correspond to the five possible combinations of the value of the Resolution Output Format parameter and the `sep` subparameter. (Note that if the Resolution

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

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

Following is the API and descriptions of the five operations.

**1. Resolve Authority to XRDS**

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

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

**2. Resolve Authority to XRD**

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

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

**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.

### 5. Resolve Service Endpoint to URI List

```
Result resolveSepToURIList(
        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 a non-empty URI List or an error as specified in section 8.2.3.
- Returns a binding-specific representation of the resolution result which may include, but not limited to, URI-list output, success/failure code, exceptions and error context.
- If successful, the output URI-list will contain zero or more elements. It is possible that the selected service contains no URI element and it is up to the consuming application to interpret such a result.

# G. Revision History

**Comment [DSR12]:** TODO-CD: We will revise this to reflect a condensed version history of XRI Resolution 2.0 as a whole.

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

|  |  |  |  |
|---|---|---|---|
|  |  |  | Added annotations to flowcharts and revised flowcharts Fig 5 and 8. |
| WD11 ED08 | 2007-11-07 | Drummond Reed | Revised text of sections 7 and 12.<br>Replaced pseudocode in section 13.6 with Wil’s compact version.<br>Replaced RelaxNG schemas with Gabe’s revised versions.<br>Replaced Appendix F with Wil’s and Steve’s revisions and editted for consistency with ED08 references and terminology. |