Extensible Resource Identifier (XRI) Resolution V2.0

Working Draft 10, 18 March 2006

Document identifier:

xri-resolution-V2.0-wd-10

Location:

http://docs.oasis-open.org/xri/xri/V2.0

Editors:

Gabe Wachob, Visa International <gwachob@visa.com>
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>

Contributors:

Dave McAlpin, Epok <dave.mcalpin@epok.net>
Chetan Sabnis, Epok <chetan.sabnis@epok.net>
Peter Davis, Neustar <peter.davis@neustar.biz>
Victor Grey, PlaNetwork <victor.grey@planetwork.org>
Mike Lindelsee, Visa International <mlindels@visa.com>

Abstract:

This document defines both generic and trusted HTTP(S)-based resolution protocols for Extensible Resource Identifiers (XRIs) as defined by *Extensible Resource Identifier (XRI) Syntax V2.0* **[XRISyntax]** or higher. For the set of XRIs defined to provide identifier metadata, see *Extensible Resource Identifier (XRI) Metadata V2.0* **[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 current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

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.

Table of Contents

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 standard protocol for resolving XRIs using HTTP(S). Both generic and trusted versions are defined (the latter using HTTPS **[RFC2818]** and/or signed SAML assertions **[SAML]**). In addition, an HTTP(S) proxy resolution version 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 data and 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 lastly 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 resolving a domain name into an attribute of a text resource descriptor using UDP, it resolves an XRI into a XML resource descriptor using HTTP(S). Table 1 provides an overview of the 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	Local, Canonical, External
Primary resolution protocol	UDP	HTTP(S)
Trusted resolution options	DNSSEC	HTTPS and/or SAML
Resolution client	local resolver	local resolver
Resolution server	authoritative nameserver	authority resolution service
Recursing resolution server	recursing nameserver	recursing authority resolution service
Proxy resolution	--	proxy resolver

Table 1: Comparing DNS and XRI resolution architecture.

As Table 1 notes, XRI resolution architecture adds one more component to the set used by DNS—a *proxy resolver*. A proxy resolver is simply a local XRI resolver with an HTTP(S) interface. Proxy resolvers enable applications—even those that do not natively understand XRIs but can process HTTP URIs—to access the functions of an XRI resolver remotely.

Figure 1 shows four typical scenarios of how these components can interact to resolve `xri://(example.root)*foo*bar` (note that unlike DNS, this works from left-to-right).

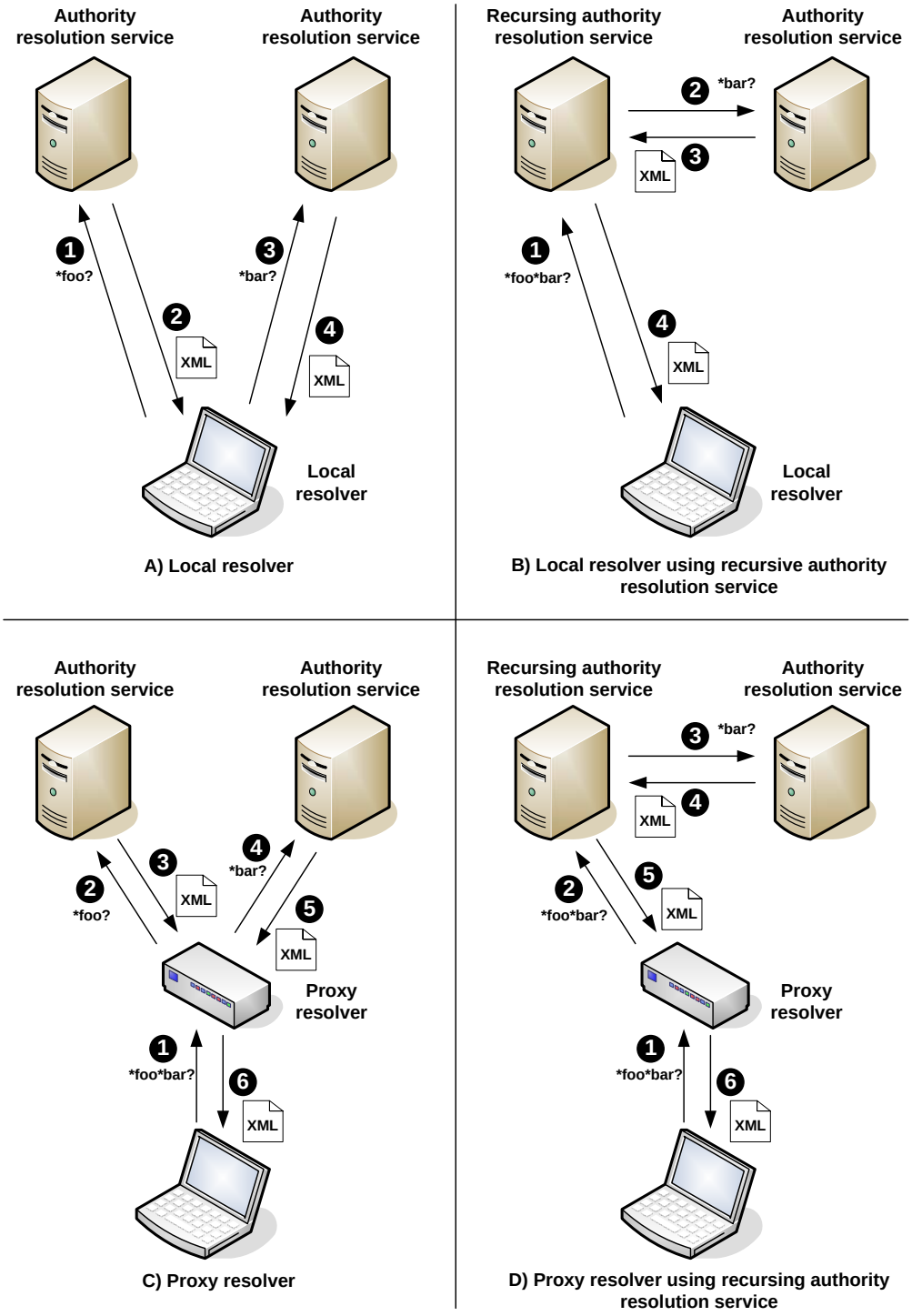

Figure 1: Four typical scenarios for XRI authority resolution.

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

- *Phase 1: authority resolution.* This is the phase required to resolve the authority segment of an XRI into the an XRDS document describing the target authority. Authority resolution works recursively from left-to-right across each subsegment in the authority segment of the XRI (subsegments are delimited using a "*" or "!" character). For example, in the XRI `xri://(example.root)*foo*bar`, the authority subsegments are `(example.root)` (the community root authority, in this case expressed as an cross-reference), `*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 never resolved.
- *Phase 2: service endpoint selection.* Once authority resolution is complete, the optional second phase of XRI resolution is to select specific metadata from the final XRDS document retreived. 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 query components of an XRI to select the service endpoint(s) to return to a calling 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 can 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 servers.
- *XRDS documents.* This simple, extensible XML resource description format makes it easy to describe the capabilities of any XRI-identified resource in a manner that can be consumed by any XML-aware application.
- *Synonyms and references.* DNS uses the CNAME attribute to establish equivalence between domain names. The XRI resolution format includes three XRI mapping elements (local IDs, canonical IDs, and external references) to provide robust support for mapping XRIs to other XRIs, IRIs, or URIs that represent the same resource. This is particularly useful for discovering and mapping persistent identifiers often required by trust infrastructures. The use of XRI references also enables multiple authorities to maintain distributed XRDS documents describing the same logical resource.
- *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 subset, type, or representation of data or metadata for the identified resource. Thus XRI resolution can serve as a lightweight, interoperable discovery mechanism for resource attributes available via LDAP, UDDI, or other directory or discovery protocols.

1.2 Structure of this Specification

This specification is structured into the following major sections:

- *Namespaces* (section 2) specifies the XRI and XML namespaces and media types used for the XRI resolution protocol.
- *XRDS Documents* (section 3) specifies a simple, flexible XML-based container for XRI resolution metadata or other metadata describing a resource.
- *Inputs and Outputs* (section 4) specifies the standard input parameters and output formats for XRI resolution.

- *Generic Authority Resolution* (section 5) specifies a simple resolution protocol for the authority segment of an XRI using HTTP/HTTPS as a transport.
- *Trusted Authority Resolution* (section 6) 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 7) 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.
- *Service Endpoint Selection* (section 8) specifies an optional second phase of resolution for selecting a set of service endpoints from an XRDS document.
- *Reference Processing* (section 9) specifies how a resolver follows XRI references to enable federation of XRDS documents across multiple XRI authorities.
- *Error Processing* (section 10) specifies error codes and error handling.
- *Use of HTTP(S)* (section 11) specifies how the XRI resolution protocol leverages features of the HTTP(S) protocol.
- *Extensibility and Versioning* (section 12) describes how the XRI resolution protocol can be easily extended and how new versions will be identified and accommodated.
- *Security and Data Protection* (section 13) summarizes key security and privacy considerations for XRI resolution infrastructure.

1.3 Examples of XRI Resolution Requests and Responses

To minimize non-normative material in the main body of the specification, extensive examples of XRI resolution requests and responses are compiled in Appendix F for generic resolution, Appendix G for SAML trusted resolution, and Appendix H for service endpoint selection.

1.4 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 **[RFC2234]**.

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.`

2 Namespaces

2.1 XRI Namespaces for XRI Resolution

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

2.1.1 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	Use	See Section
xri://$res	Namespace for XRI resolution service types	2.1.2
xri://$xrds	Namespace for the generic XRDS (Extensible Resource Descriptor Sequence) schema (not versioned)	2.2
xri://$xrd	Namespace for the XRD (Extensible Resource Descriptor) schema	2.2
xri://$xrd*($v*2.0)	Version 2.0 of above (using XRI version metadata as defined in **[XRIMetadata]**)	2.2

Table 2: XRIs reserved for XRI resolution.

2.1.2 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	Use	See Section
xri://$res*auth	Authority resolution service	5
xri://$res*auth*($v*2.0)	Version 2.0 of above	5
xri://$res*proxy	HTTP(S) proxy resolution service	7
xri://$res*proxy*($v*2.0)	Version 2.0 of above	7

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.

2.2 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://$xrds	Section 2.1.1 of this document
xrd	xri://$xrd*($v*2.0)	Section 2.1.1 of this document

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

310 2.3 Media Types for XRI Resolution

311 Because XRI resolution architecture is based on HTTP, it makes use of standard media types as
312 defined by **[RFC2046]**, particularly in HTTP Accept headers as specified in **[RFC2616]**. Table 5
313 specifies the media types used for XRI resolution.

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 8	Section 5 of **[RFC2483]**

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

315 To provide full control of XRI resolution via an HTTP interface, these media types accept media
316 types parameters as defined in Table 6.

Parameter	Values	Applies to Media Type	Usage
trust	none https saml https+saml	application/xrds+xml application/xrd+xml text/uri-list	Specifies use of a trusted resolution protocol (section 6)
refs	true false	application/xrds+xml application/xrd+xml text/uri-list	Specifies whether references should be followed during resolution (section 9)
sep	true false	application/xrds+xml application/xrd+xml	Specifies whether service endpoint selection should be performed (section 8)

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

318 See sections 4 - 8 for more about usage of these media types and parameters.

3 XRDS Documents

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

3.1 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 a single attribute, `xrds:XRDS/@xrds:ref` of type `anyURI` that identifies the resource described by the XRDS document. The formal XML schema definition of an XRDS document is provided in Appendix A.

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]**.

This namespace architecture enables the XRDS namespace to remain constant while the XRD namespace (and the namespaces of other XML elements that may be included in an XRDS document) may be versioned over time. See section 12.2 for more about versioning of the XRD schema.

3.2 XRD Elements and Attributes

The following example XRDS instance document illustrates the elements and attributes defined in the XRD schema:

```
<XRDS xmlns="xri://$xrds" ref="xri://(example.root)*foo">
    <XRD xmlns="xri://$xrd*($v*2.0)" version=”2.0”>
        <Query>*foo</Query>
        <Status code="100"/>
        <Expires>2005-05-30T09:30:10Z</Expires>
        <ProviderID>urn:uuid:c9f812f3-6544-4e3c-874e-
d3ae79f4ef7b</ProviderID>
        <LocalID>*baz</LocalID>
        <CanonicalID>xri://(example.root)!1234!5678</CanonicalID>
        <Ref>xri://!!4A76!C2F7!9033</Ref>
        <Service>
            <ProviderID>xri://!!1000!1234.5678</ProviderID>
            <Type>xri://$res*auth*($v*2.0)</Type>
            <MediaType>application/xrds+xml</MediaType>
            <URI priority=”10”>http://resolve.example.com</URI>
            <URI priority=”15”>http://resolve2.example.com</URI>
            <URI>https://resolve.example.com</URI>
        </Service>
        <Service>
            <ProviderID>xri://!!1000!1234.5678</ProviderID>
            <Type>xri://$res*auth*($v*2.0)</Type>
            <MediaType>application/xrds+xml;trust=https</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 XML schema definition of the XRD schema is provided in Appendix A. 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.

3.2.1 Management Elements

The first set of elements are used to manage XRDs, especially from the perspective of caching and error handling.

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 `xrd:XRDS` document. It also includes an optional `xrd: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 3.3 and 9.2. Lastly, it includes an `xrd:version` attribute that is optional for uses outside of XRI resolution but REQUIRED for XRI resolution as defined in section 3.3.2

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. For XRI authority resolution, this must be a qualified subsegment of the authority component of the query XRI.

xrd:XRD/xrd:Status

0 or 1 per `xrd:XRD` element. Contains a required attribute `xrd:code` of type `xs:int` that provides a numeric status code. The contents of the element are an optional human-readable message string describing the status of the response. For XRI resolution, values of the Status element and `xrd:code` attribute are defined in section 10.

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 using this XRD MUST NOT use the 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 11.2.1.

3.2.2 Authority Trust Elements

The second set of elements are for applications where trust must be established in the authority providing the XRD. These elements are OPTIONAL for generic authority resolution (section 5), but REQUIRED for trusted authority resolution (section 6).

xrd:XRD/xrd:ProviderID

0 or 1 per `xrd:XRD`. A unique identifier of type `xs:anyURI` for the authority producing this XRD. There MUST be negligible probability that the value of this element will be assigned as an identifier to any other authority. This element MUST be a persistent identifier such as a URN **[RFC2141]** or a fully persistent XRI **[XRISyntax]**. Note that for XRI authority resolution, the authority identified by this element is the *describing* authority (the producer of the current XRD), not the authority *described* by the XRD. The latter is specified in the `xrd:XRD/xrd:Service/xrd:ProviderID` element for a resolution service endpoint (see below).

xrd:XRD/saml:Assertion

0 or 1 per `xrd:XRD`. A SAML assertion from the *describing* authority (the one producing 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 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 resolution root to place restrictions, if any, on the `saml:Issuer` element. A suitable approach is to use an XRI in URI-normal form that describes the community root authority. See section 5.1.3.

3.2.3 Synonym Elements

A third set of elements are used to provide *synonyms*—identifiers that are not character-for-character equivalent to the contents of the Query element, but which identify the same target resource. For more information about synonyms in XRI resolution, see section 9.1. Because there may be multiple instances of each type of synonym, all three of these elements have the global `xrd:priority` attribute (see section 3.3.3).

xrd:XRD/xrd:LocalID

0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST NOT be an absolute identifier. MUST be an interchangeable synonym for the contents of the `xrd:XRD/xrd:Query` element, i.e., an XRI, IRI, or URI reference in URI normal form assigned to the same target resource by the same authority producing the current XRD. It MUST resolve to the current XRD (with the exception of the `xrd:Query`, `xrd:Expires`, and `xrd:LocalID` elements.)

xrd:XRD/xrd:CanonicalID

0 or more per `xrd:XRD` element. Type `xs:anyURI`. MUST be an absolute identifier. MUST contain an absolute XRI, IRI, or URI that serves as a canonical identifier for the described resource, i.e. the preferred synonym among all synonyms. If the contents is an XRI, it may or may not be resolvable and if resolvable may or may not resolve to the same XRD. An XRD element SHOULD contain only one instance of a CanonicalID element, however in some circumstances (such as the merging of two previously distinct resources) this may not be possible. Applications using this element to identify the target resource SHOULD try alternate instances if the top priority instance is not recognized.

xrd:XRD/xrd:Ref

0 or more per `xrd:XRD` element. MUST be an absolute identifier. Identical to `xrd:XRD/xrd:CanonicalID` except this identifier MUST be assigned by a different authority than the authority producing the current XRD. Resolution of this reference MAY produce a different XRD for the described resource. See section 9 for complete details of reference processing.

3.2.4 Service Endpoint Elements

The next set of elements are used to describe service endpoints—the set of network endpoints advertised in an XRD for performing further resolution, obtaining further metadata, or interacting directly with the described resource. Again, because there can be more than one instance of a particular type of service endpoint that satisfies a service endpoint selection query, or more than one instance of a service endpoint URI, these elements both have the global `xrd:priority` attribute (see section 3.3.3).

xrd:XRD/xrd:Service

0 or more per `xrd:XRD` element. The container element for service endpoint metadata.

xrd:XRD/xrd:Service/xrd:URI

0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:anyURI`. If present, it indicates a transport-level URI for access to the capability described by the parent Service element. For the XRI resolution service types defined in section 2.1.2, this URI MUST be an HTTP or HTTPS URI. Other services may use other transport protocols. It includes an `xrd:append` attribute that governs construction of the final service endpoint URI. See section 8.4.

3.2.5 Service Endpoint Trust Elements

Similar to the authority trust elements, these elements enable trust to be established in the provider of the service endpoint. These elements are OPTIONAL for generic authority resolution (section 5), but REQUIRED for trusted authority resolution (section 6).

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 trusted resolution, when a resolution request is made to the 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. See section 6.2.5. The same usage MAY apply to other services not defined in this specification.

xrd:XRD/xrd:Service/ds:KeyInfo

0 or 1 per `xrd:XRD/xrd:Service` element. Provides the digital signature metadata necessary to validate an XRD provided as a resolution response by the described authority. This element comprises the key distribution method for trusted authority resolution in the XRI resolution framework—see section 6.2.5.

3.2.6 Service Endpoint Selection Elements

The final set of elements are used in XRI resolution to select service endpoints for further processing. They include two global attributes used for this purpose: `xrd:match` and `xrd:select`. See sections 8.2 and 8.3.

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 2.1.2 for the resolution service types defined in this specification.

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 2.3 for the media types used in XRI resolution.

xrd:XRD/xrd:Service/xrd:Path

0 or more per `xrd:XRD/xrd:Service` element. Of type `xs:string`. Contains a string value meeting the xri-path production defined in section 2.2.3 of **[XRISyntax]**.

The XRD schema (Appendix A) allows other elements and attributes from other namespaces to be added throughout. As described in section 12.1.1, these points of extensibility can be used to deploy new XRI resolution schemes, new service description schemes, or other metadata about the described resource.

3.3 XRD Attribute Processing Rules

3.3.1 ID Attribute

For uses such as XRI trusted resolution (section 6.2) 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 resolution services 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 5.1.2. 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 9.

3.3.2 Version Attribute

Unlike the XRDS element, which is not intended to be versioned, the `xrd:XRD` element has the optional attribute `xrd:XRD/@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 its containing XRD element conforms. See section 2.1.1.

For more about versioning of the XRI resolution protocol, see section 12.2.

3.3.3 Priority Attribute

Certain XRD elements involved in the XRI resolution process (`xrd:Ref`, `xrd:Service`, and `xrd:URI`) may be present multiple times in an XRDS document to describe different references, redundant service endpoints, or for other reasons. 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 client 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 “15”. Zero is the highest priority attribute value. Null is the lowest priority attribute value.
2. If an element has no priority attribute, its priority attribute value is considered to be null.
3. If two or more instances of the same element type have identical priority attribute values (including the null value), the client SHOULD select one of the instances at random. This client SHOULD NOT simply choose the first instance that appears in XML document order (this is important in order to support intentional load balancing).
4. An element selected according to these rules is referred to in this specification as “the highest priority element”. If this element is subsequently disqualified from the set of qualified elements, the next element selected according to these rules is referred to as “the next highest priority element”. If an XRI 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.

When setting priority attributes, it is recommended that XRI authorities follow the standard practice in DNS and set the default highest priority attribute value to “10”.

3.4 XRI 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 used in XRI resolution because it relies on HTTP(S) as a transport protocol. When an XRI is used as the value of an `xrd:Query`, `xrd:LocalID`, `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 that XRIs composed entirely of valid URI characters and that do not use XRI 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 cross-reference syntax may require percent encoding in the transformation to URI-normal form as explained in section 2.3 of **[XRISyntax]**.

4 Inputs and Outputs

Logically, XRI resolution is a function invoked by an application to dereference an XRI into a media type that describes the target resource. This section defines the logical inputs and outputs of this function, however it does not specify any particular binding to any local resolver interface. For purposes of illustration, a non-normative, language-neutral API is suggested in Appendix C.

A binding to an HTTP interface for XRI proxy resolvers is specified in section 7.

4.1 Inputs

Table 7 summarizes the logical input parameters to the authority phase of XRI resolution (section 5). In this specification, references to these parameters will use the names in the first column. Local APIs MAY use different names for these parameters and MAY define additional parameters.

Logical Parameter Name	Type	Required/ Optional	Default
QXRI (query XRI)	xs:anyURI	Required	N/A
Resolution Media Type	xs:string (media type)	Optional	Null

Table 7: Input parameters for the authority phase of XRI resolution.

Table 8 summarizes the additional input parameters used in the service endpoint selection phase of XRI resolution (section 7).

Logical Parameter Name	Type	Required/ Optional	Default
Service Type	xs:anyURI	Optional	Null
Service Media Type	xs:string (media type)	Optional	Null

Table 8: Input parameters for the service endpoint selection phase of XRI resolution.

The following sections specify additional validation and usage requirements.

4.1.1 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 leading double forward slashes ("//") or terminating single forward slash ("/").
Path String	xs:string	Optional	Contents of the path component of the QXRI, not including 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 normal forms but SHOULD document the normal form(s) it supports and its normalization policies.

4.1.2 Resolution Media Type

The Resolution Media Type 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 references should be followed during resolution.
- Whether final service endpoint selection should be performed.

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

1. The value of Resolution Media Type 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 `trust` media type parameter is `none` or null, or if this parameter is absent, the resolver MAY use its choice of authority resolution protocol.

3. If the value of the `trust` media type parameter is `https`, the resolver MUST use the HTTPS trusted authority resolution protocol specified in section 6.1 (or return an error saying this is not supported).
4. If the value of the `trust` media type parameter is `saml`, the resolver MUST use the SAML trusted authority resolution protocol specified in section 6.2 (or return an error saying this is not supported).
5. If the value of the `trust` media type parameter is `https+saml`, the resolver MUST use the HTTPS+SAML trusted authority resolution protocol specified in section 6.3 (or return an error saying this is not supported).
6. If the value of the `refs` media type parameter is `true` or null, or if the parameter is absent, the resolver MUST perform reference processing as defined in section 9 if it is necessary to complete resolution (or return an error saying this is not supported).
7. If the value of the `refs` media type parameter is `false`, the resolver MUST NOT perform reference processing during resolution.
8. If the value of the `sep` media type parameter is `true`, the resolver MUST perform final service endpoint selection (or return an error saying this is not supported).
9. If the value of the `sep` media type parameter is `false`, the resolver MUST NOT perform service endpoint selection.
10. If the `sep` media type parameter is absent or null, the resolver MUST NOT perform service endpoint selection if the value of Resolution Media Type is either `application/xrds+xml` or `application/xrd+xml`, but MUST perform service endpoint selection if the value of Resolution Media Type is any other value, including null.

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

4.1.3 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 7). 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 specified so that this parameter may be passed to a proxy resolver in a QXRI query parameter as defined in section 7.) The Service Type values defined for XRI resolution services are specified in section 2.1.2.

4.1.4 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 7). 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 2.3.

4.2 Outputs

Table 10 summarizes the logical outputs of XRI resolution.

Logical Output Name	Media Type Value for Requesting Authority Resolution Only	Media Type Value for Requesting Final Service Endpoint Selection
XRDS Document	application/xrds+xml	application/xrds+xml;sep=true
XRD Document	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.

The following sections provide additional construction and validation requirements.

4.2.1 XRDS Document

If the value of the Resolution Media Type 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. Also, any nested XRDS documents included as a result of reference processing must also be valid.
2. Each of the contained XRD elements must be a valid XRD document according to the schema defined in Appendix A.
3. The XRD elements MUST conform to the additional requirements in section 3.
4. If the value of the `trust` media type parameter is `saml` or `https+saml`, the XRD element MUST further conform to the additional requirements in section 6.2.
5. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST be performed as defined in section 8, even if the values of all three service endpoint selection input parameters (Service Type, Service Media Type, and Path String) are null.
6. If reference processing is necessary during the service endpoint selection process, the final child element of the root level `xrds:XRDS` element MUST be a nested XRDS document as specified in section 9. Any other filtering of the final XRD element MUST NOT be performed.
7. 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 10.

4.2.2 XRD Document

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

1. The output MUST be a valid XRD document according to the schema defined in Appendix A.
2. The XRD elements MUST conform to the additional requirements in section 3.
3. 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 document MUST NOT be performed.

4. If the value of the `sep` media type parameter is `true`, service endpoint selection MUST be performed as defined in section 8, even if the values of all three service endpoint selection input parameters (Service Type, Service Media Type, and Path String) are null.
5. If service endpoint selection is performed, the XRD document MUST only include the `xrd:Service` elements selected according to the rules specified in section 8 (or return an error if no `xrd:Service` elements were selected). In addition, all XRD elements that are subject to the global `xrd:priority` attribute (even if the attribute is absent or null) MUST be returned in order of highest to lowest priority as defined in section 3.3.3. Any other filtering of the XRD document MUST NOT be performed.
6. If the output is an error, this error MUST be returned using the `xrd:Status` element as defined in section 10.

4.2.3 URI List

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

1. For this output, service endpoint selection is REQUIRED.
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 3.3.3.
4. From the final selected `xrd:XRD/xrd:Service` element, the service endpoint URI(s) MUST be constructed as defined in section 8.4.
5. 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 3.3.3. 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.
6. If the output is an error, it MUST be returned with the content type `text/plain` as defined in section 10.

4.2.4 HTTP(S) Redirect

In XRI proxy resolution, the Resolution Media Type parameter may be null. In this case the output of a proxy resolver is an HTTP(S) redirect as defined in section 7.6.

5 Generic Authority Resolution

Authority resolution is the first phase of XRI resolution as described in section 1.1. It applies only to the Authority String of the QXRI. This may be 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 5.1. 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 5.2.

5.1 XRI Authority Resolution

5.1.1 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	trust=none (this is the default – see 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 Service Media Type identifier defined in Table 11. Because `trust=none` is the default value of trust parameter, the values `application/xrds+xml` and `application/xrds+xml;trust=none` MUST be considered equivalent for the purposes of both service endpoint selection and HTTP(S) Accept header values as described below.

5.1.2 Protocol

Figure 2 shows the overall logical flow of the generic authority resolution process.

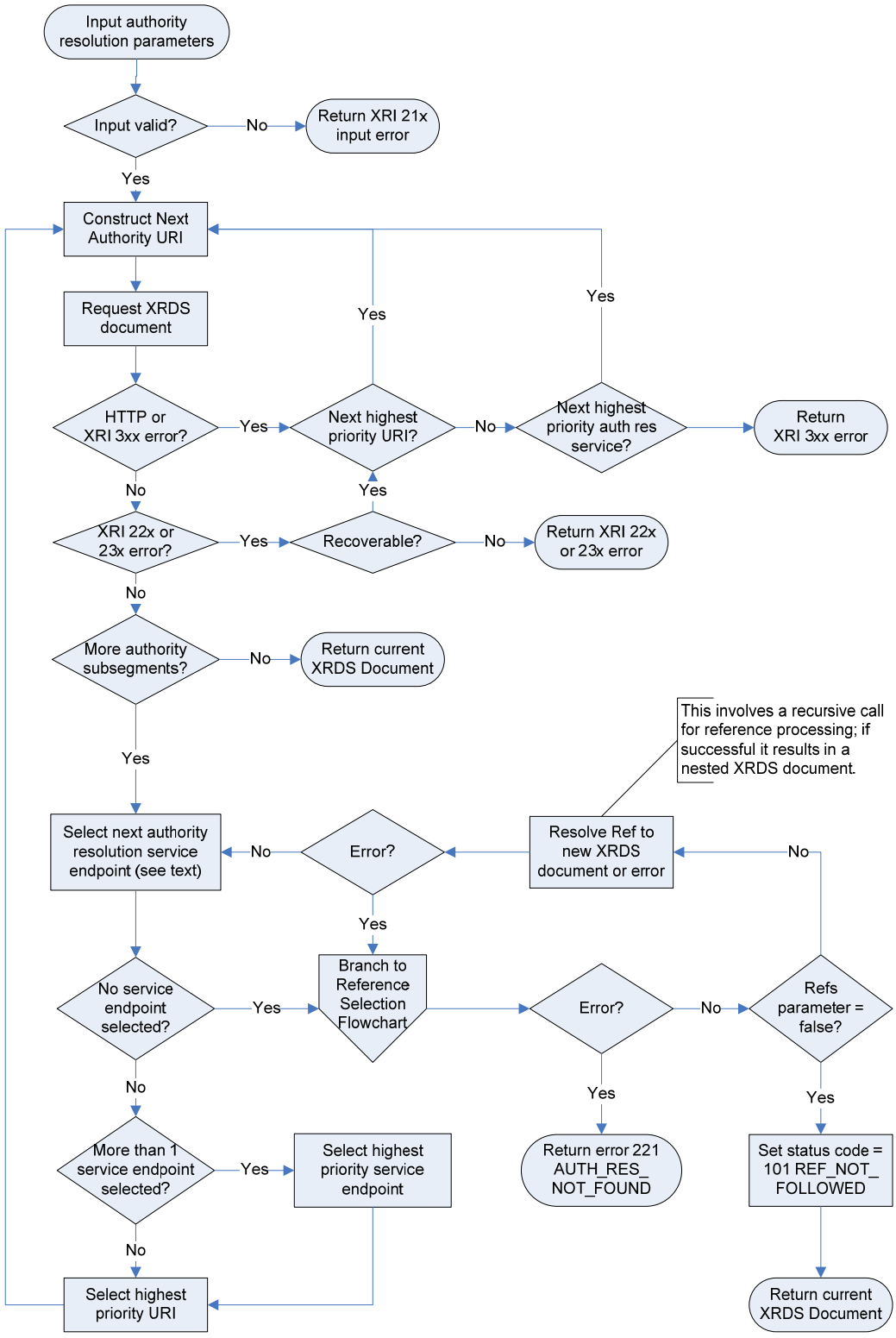

Figure 2: Authority resolution flowchart

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

1. The resolver MUST be preconfigured with (or have a means of obtaining) the XRD document describing the community root authority for the XRI to be resolved as defined in section 5.1.3.
2. 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 5.1.4. If the final output is an XRDS document, this 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. In addition, any references followed MUST be represented by nested `xrd:XRDS` documents immediately following the `xrd:XRD` element containing the reference as defined in section 9.
3. Subsegments that use XRI cross-reference syntax MUST be resolved as defined in section 5.1.5.
4. The resolver MAY request that a recursing authority resolution service perform resolution of multiple subsegments as defined in section 5.1.6.
5. For each iteration of the authority resolution process, the next authority resolution service endpoint MUST be selected as defined in section 7 using the values from Table 11. In addition, the Next Authority URI MUST be constructed as defined in section 5.1.7.
6. Each authority subsegment MUST be resolved via an HTTP(S) GET request to the Next Authority URI.
7. The HTTP(S) request MUST contain an Accept header with the media type identifier defined in Table 11. This media type identifier MUST be interpreted as the value of the Resolution Media Type input parameter.
8. The ultimate HTTP(S) response from an authority resolution service 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]**.
9. Any ultimate response besides an HTTP 2XX or 304 SHOULD be considered an error in the resolution process and the resolver SHOULD return the appropriate error code and context message as specified in section 10. In recursive resolution, such an error MUST be returned by the recursing authority resolution service to the resolver as specified in section 10.3.
10. If an XRD does not include the next requested authority service endpoint but includes one or more `xrd:Ref` elements, the resolver MUST perform reference processing as defined in section 9 unless the `refs` media type parameter defined in Table 6 is set to `false`. If the `refs` media type parameter is set to `false` and the XRD contains at least one `xrd:Ref` element that could be followed, the resolver MUST return a successful response with a status code of `101 REF_NOT_FOLLOWED`. (Note that such reference processing, if successful, will result in a separate nested XRDS document describing the resolved reference.)
11. A successful response that does not include the next required authority service endpoint in the XRD and does not include any `xrd:Ref` elements MUST return an error with a status code of `221 AUTH_RES_NOT_FOUND` regardless of the value of the `refs` media type parameter.

12. All other uses of HTTP(S) in this protocol MUST comply with the requirements in section 11. 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 11.2.1.

5.1.3 Community Root Authorities

Identifier management policies are defined on a community-by-community basis. For XRI 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*. 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.

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 an XRDS document at a known location simplifies this process. It is also a recommended best practice for this 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 XRI Technical Committee home page at http://www.oasis-open.org/committees/xri.

5.1.4 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 5.1.7. 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.

5.1.5 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.

5.1.6 Recursing Authority Resolution

If an authority resolution service 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 resolution service MAY perform the additional recursing resolution steps requested. In this case the recursing authority resolution service acts as a resolver to the other authority resolution service endpoints that need to be queried. Alternatively, the recursing authority resolution service 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 resolution service performs any recursing resolution, it MUST return an ordered list of `xrd:XRD` elements (and nested `xrd:XRDS` elements if references are followed) in an `xrd:XRDS` document for all subsegments resolved as defined in section 5.1.2.

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

If the recursing authority resolution service does not resolve the entire set of subsegments requested, the resolver is responsible for continuing the authority resolution process itself. At any stage, however, the resolver MAY request that the next authority resolution service recursively resolve any remaining subsegments.

5.1.7 Construction of the Next Authority URI

At each step in authority resolution, a URI must be constructed for the next HTTP(S) request. This URI is constructed by the XRI resolver from two strings—one representing the next authority resolution service endpoint selected from the current XRD, and the other representing the next unresolved subsegment or group of subsegments in the QXRI Authority String.

The process for selecting the next authority resolution service endpoint from the current XRD is defined in section 8. For generic authority resolution, the process MAY select the this service endpoint from the parameters defined in Table 11, or from any of the parameters defined for selection of trusted resolution service endpoints in section 6.

From the output of the service endpoint selection process, the resolver MUST select the highest priority URI of the highest priority authority resolution service endpoint. This is called the *Next Authority Service Endpoint URI*. If this URI does not end with a forward slash ("/"), one MUST be appended before proceeding.

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

- The next fully qualified subsegment to be resolved (see section 5.1.4), or
- In the case of recursing resolution, the next fully qualified subsegment plus any additional subsegments for which recursing resolution is requested (see section 5.1.5).

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 pseudo-code 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
```

5.2 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.

5.2.1 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.

5.2.2 Protocol

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

1. The next authority URI is constructed by extracting the entire IRI authority segment and prepending the string "http://". See the exception in section 5.2.3.
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.
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:

```
Host: example.com
```

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.

5.2.3 Optional Use of HTTPS

Section 6 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 6.1.

6 Trusted Authority Resolution

This section defines three options for performing trusted XRI authority resolution as an extension of the generic XRI authority resolution service defined in section 5.1—one using HTTPS, one using SAML assertions, and one using both.

6.1 HTTPS

This option for trusted authority resolution is a very simple addition 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.

6.1.1 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	trust=https

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 `trust=https` 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.

6.1.2 Protocol

Following are the normative requirements for HTTPS trusted authority resolution that differ from generic XRI authority resolution (section 5.1):

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]**. 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 (including the `trust=https` parameter) defined in Table 15. This MUST be interpreted as the value of the Resolution Media Type input parameter.
4. If the resolver finds that an authority in the resolution chain does not support HTTPS, the resolver MUST return a 23x error as defined in section 10.

6.2 SAML

In SAML trusted resolution, the resolver requests a content type of `application/xrds+xml;trust=saml` and the authority resolution service 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 it

with HTTPS as defined in section 6.3. 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.

6.2.1 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	trust=saml

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

A SAML trusted resolution service endpoint advertised in an XRD document MUST use the Service Type identifier and Service Media Type identifier (including the `trust=saml` parameter) defined in Table 16. 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.

6.2.2 Protocol

6.2.2.1 Client Requirements

For a resolver, trusted resolution is identical to the generic resolution protocol (section 5.1) 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 (including the `trust=saml` parameter) defined in Table 16. This MUST be interpreted as the value of the Resolution Media Type input parameter. (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 is NOT REQUIRED 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 6.2.3.
5. The resolver MUST individually validate each XRD in the resolution chain according to the rules defined in section 6.2.4. When `xrd:XRD` elements come both from freshly-retrieved XRD documents and from a local cache, a resolver MUST ensure that these requirements are satisfied each time a resolution request is performed.

6.2.2.2 Server Requirements

For an authority resolution service, trusted resolution is identical to the generic resolution protocol (section 5.1) 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;trust=saml`".

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 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. In **[SAML]**, 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. Because the signing key is known in advance by the resolution client, the `ds:KeyInfo` element SHOULD be omitted from the digital signature.
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 6.2.5.
8. The `xrd:XRD/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.

6.2.3 Recursing Authority Resolution

If a resolver requests trusted resolution of multiple authority subsegments (see section 5.1.5), a recursing authority resolution service 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.

6.2.4 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 6.2.5.
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 6.2.5.
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.

6.2.5 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 URNs as defined by **[RFC2141]** and fully persistent XRIs converted to URI-normal form as defined by **[XRISyntax]**.

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 the community root XRDS document (or other equivalent description—see section 5.1.3) 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 3.2.) 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 resolution services 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]**.

6.3 HTTPS+SAML

6.3.1 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	trust=https+saml

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 (including the `trust=https+saml` parameter) defined in Table 17. In addition, the identifier authority MUST use an HTTPS URI as the value of the `xrd:URI` element(s) for this service endpoint.

6.3.2 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 6.1 and the SAML trusted resolution protocol defined in section 6.2.
3. All authority resolution requests MUST contain an HTTPS Accept header with the media type identifier (including the `trust=https+saml` parameter) defined in Table 17. This MUST be interpreted as the value of the Resolution Media Type 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 10.

7 Proxy Resolution

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 includes a standard syntax for expressing an XRI as an HTTP URI, called an *HXRI*, and 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 11.4.
- 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.

7.1 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 application/xrd+xml text/uri-list	trust=none trust=https trust=saml trust=https+saml refs=true refs=false sep=true sep=false

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 (including the optional parameters) defined in Table 18. In addition, if the media type parameters `trust=https` or `trust=https+saml` is included, the identifier authority 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 (section 5.1.3) 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.

7.2 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 "`xri.`". The QXRI is then appended as the entire local path (and query component, if present). In essence, the proxy resolver URI including the forward slash after the domain name serves as a machine-readable prefix for an absolute XRI.

Following is the normative ABNF for an HXRI is defined below based on the `XRI` and `ireg-name` productions defined in **[XRISyntax]**. Authors of XRIs that 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

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

QXRI              = XRI
```

URI processors that recognize XRIs SHOULD interpret the local part of any HTTP or HTTPS URI that conforms to this ABNF as an XRI provided the domain name is at least three levels deep (e.g., "xri.example.com".) If the URI conforms to this ABNF but the domain name is only two levels deep, URI processors SHOULD interpret the local part as an XRI only if the second-level domain is known to offer XRI proxy resolution services.

For references to communities that offer public XRI proxy resolution services, see the XRI Technical Committee home page at http://www.oasis-open.org/committees/xri.

7.3 QXRI Query Parameters

Besides the QXRI, there are three other logical input parameters to XRI authority resolution and service endpoint selection as defined in section 4.1: Resolution Media Type, Service Type, and Service Media Type. These parameters are bound to an HTTP(S) interface using the conventional web model of passing parameters in the query component of the HTTP URI (which in the case of an HXRI means the query component of the QXRI). The binding of the logical parameter names to QXRI query parameter names is defined in Table 19.

Logical Parameter Name	QXRI Query Parameter Name
Resolution Media Type	_xrd_r
Service Type	_xrd_t
Service Media Type	_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. If the original QXRI has an existing query component, these parameters MUST be added 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.) If the original QXRI does not have a query component, one MUST be added to pass these parameters. 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 when adding any XRI resolution parameters. Any XRI query parameters and question marks will later be removed by the proxy resolver in the URI construction step defined in section 8.4.
2. Each parameter MUST be delimited from other parameters by an ampersand ("&"). Any occurrences of the character "&" within an input parameter (specifically the Service Type value) MUST be percent encoded prior to input.
3. Each parameter MUST be delimited from its value by an equals sign ("=").
4. If any parameter name is included but its value is empty, the value of the parameter MUST be considered null.
5. For proxy resolution, any input parameter supplied explicitly via a QXRI query parameter MUST take precedence over the same parameter provided via any other interface, even if the value of the QXRI-supplied parameter is explicitly null. See the following section.

7.4 HTTP(S) Accept Headers

XRI resolution input parameters MAY also be passed to a proxy resolver via the HTTP(S) Accept header of a resolution request. These will take the form of the media type identifiers and associated parameters defined in Table 18. Following are the rules governing the use of such inputs in XRI proxy resolution.

1. If the Accept header consists of multiple media type identifiers and their associated parameters as described in section 14.1 of **[RFC2616]**, 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 SHOULD choose the first one it supports.
2. If a media type identifier matches one of the Service Types in Table 18, it MUST be interpreted as the implicit value of the Resolution Media Type parameter, and the implicit value of the Service Media Type parameter MUST be null.
3. If a media type identifier does not match one of the Service Types in Table 18, it MUST be interpreted as the implicit value of the Service Media Type parameter, and the implicit value of the Resolution Media Type parameter MUST be null.
4. If the value of the Accept header content type is null, then this MUST be interpreted as the implicit value of both the Resolution Media Type and Service Media Type parameters (i.e., both are set to null).
5. In all cases, if the value of either the Resolution Media Type or Service Media Type parameters is explicitly set via the `_xrd_r` or `_xrd_m` query parameters in the QXRI (including to a null value), this explicit value MUST override any implicit value that may be set via an HTTP(S) Accept header.
6. All other rules for processing of the Resolution Media Type input parameter apply from section 4.1.2. In particular, the `refs` parameter allows proxy resolver operators to control whether their local resolver will perform reference processing, and the `sep` parameter allows proxy resolver clients to control whether they want the proxy resolver to perform service endpoint selection.

7.5 Null Resolution Media Type

Unlike authority resolution as defined in the preceding sections, a proxy resolver MAY receive a resolution request where the Resolution Media Type input parameter value is null—either because this parameter was not implicitly set using an HTTP Accept header media type identifier as defined in the previous section or because it was explicitly set to null using the `_xrd_r` query parameter.

If the value of the Resolution Media Type value is null, the values of its media type parameters MUST be set to `trust=none`, `refs=true`, and `sep=true`. In addition, the output MUST be an HTTP(S) redirect as defined in the following section.

7.6 Outputs and HTTP(S) Redirects

For all values of the Resolution Media Type parameter except null, a proxy resolver MUST follow the output rules defined in section 4.2.

If the value of the Resolution Media Type 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 4.2.3. 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 with a media type of either `text/plain` or `text/html`.

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).

7.7 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 attempt to deliver uniform responses given the same QXRI and 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.

8 Service Endpoint Selection

If the authority resolution phase is successful, the output is an XRDS document containing a final XRD describing the target authority. If requested, a resolver may perform an optional second phase of XRI resolution by processing this XRDS document to select a requested service endpoint. This section specifies the rules for this process.

8.1 Processing Rules

Figure 3 shows the overall logical flow of the service endpoint selection process.

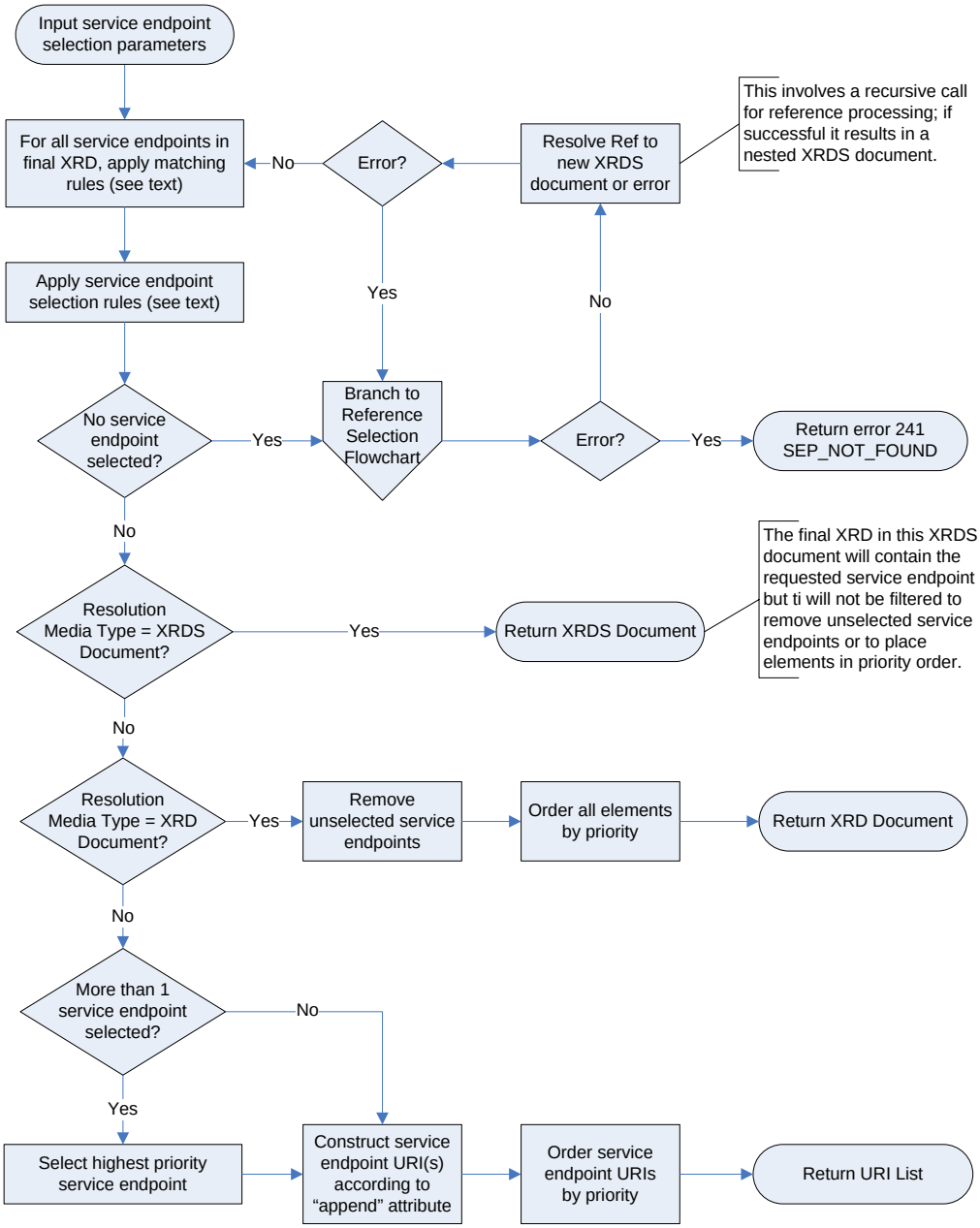

Figure 3: Service endpoint selection flowchart.

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

1. The inputs for service endpoint selection include both those defined for authority resolution (Table 7) plus those defined only for service endpoint selection (Table 8).
2. For each `xrd:Service` element in an XRD, selection is controlled by matching its three child service endpoint selection elements (`xrd:Type`, `xrd:MediaType`, and `xrd:Path`) to the values of the corresponding the Service Type, Service Media Type, and Path String parameters. This matching process MUST follow the rules defined in section 8.2.
3. Following the matching process, selection of `xrd:Service` elements MUST follow the selection rules defined in section 8.3.
4. If after applying the selection rules no service endpoint is selected, a resolver MUST perform reference processing as defined in section 9 if the `refs` media type parameter is set to `true` or null, or if the parameter is absent. If the `refs` media type parameter is set to `false` and the XRD contains at least one `xrd:Ref` element that could be followed, the resolver MUST return a successful response with a status code of `101 REF_NOT_FOLLOWED`.
5. If an XRD does not include the requested service endpoint and does not include any `xrd:Ref` elements, the resolver MUST return an error with a status code of `241 SEP_NOT_FOUND` regardless of the value of the `refs` media type parameter.
6. The output of service endpoint selection MUST be a valid instance of the media type value of the Resolution Media Type parameter. If this output is an XRDS Document, an XRD Document, or a URI List, it MUST conform to the output requirements defined in section 4.2. If the Resolution Media Type value is null and the output is an HTTP(S) redirect, the resolver MUST proceed as defined in section 7.6.

8.2 Matching Rules

Matching of an `xrd:Service` element is controlled by its three service endpoint selection elements, `xrd:Type`, `xrd:MediaType`, and `xrd:Path`, and their optional `xrd:match` attribute. Note that the rules in this section only determine *whether there is a match on the service endpoint selection elements*. The rules for selecting the parent `xrd:Service` element on the basis of these matches are defined in section 8.3. For examples, see Appendix H.

8.2.1 Match Attribute Values

The `xrd:match` attribute accepts an enumerated list of values as defined in Table 20.

Value	Matching Rule (for corresponding input parameter)
default	Match this element only if there are no other matches in the XRD on this element type (except another default match). This is the default value if an instance of an element that accepts the `xrd:match` attribute is *entirely* absent.
content	Match the contents of this element. This is the default value if an element that accepts the `xrd:match` attribute is present but the attribute is omitted or its value is null.
any	Match any value (null or non-null).
non-null	Match any value except null.
null	Match a null value.
none	Do not match. The containing `xrd:Service` element is not currently active.

Table 20: Enumerated values of the global match attribute and corresponding matching rules.

The following matching rules apply based on the value of the `xrd:match` attribute:

1. If an `xrd:Type`, `xrd:MediaType`, or `xrd:Path` element is omitted, it MUST be considered equivalent to including an instance of the respective element with an `xrd:match` attribute value of `default`.
2. If an `xrd:Type`, `xrd:MediaType`, or `xrd:Path` element is present but its contents are empty, the value of the element MUST be considered null.
3. If an `xrd:Type`, `xrd:MediaType`, or `xrd:Path` element is present but the `xrd:match` attribute is omitted or has the default value of `content`, the respective content matching rules in the following three sections MUST be applied.
4. For all other values of the `xrd:match` attribute, including `default`, `any`, `non-null`, `null`, and `none`, the element contents MUST be ignored and the matching rules in Table 20 MUST be applied.

8.2.2 Service Type Matching

The following rules apply when the `xrd:match` attribute of an `xrd:Type` element is omitted or has the default value of “content”.

1. The values of the Service Type parameter and the `xrd:Type` element SHOULD be normalized according to the requirements of their identifier scheme prior to input. In addition, if the value is an XRI or an IRI it MUST be in URI-normal form as defined in section 2.3 of **[XRISyntax]**. XRI resolvers MAY perform normalization of these values but MUST NOT be required to do so.
2. To match, the values MUST be equivalent according to the equivalence rules of the applicable identifier scheme.

8.2.3 Service Media Type Matching

The following rules apply when the `xrd:match` attribute of an `xrd:MediaType` element is omitted or has the default value of “content”.

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 media type parameters are case-insensitive.) XRI resolvers MAY perform normalization of these values but MUST NOT be required to do so.
2. To match, the values MUST be character-for-character equivalent.

8.2.4 Path String Matching

The following rules apply when the `xrd:match` attribute of an `xrd:Path` element is omitted or has the default value of “content”.

1. The values of the Path String parameter and the `xrd:Path` element MUST be normalized to remove any trailing space or XRI delimiter (“/”, “*”, or “!”).
2. Equivalence comparison MUST be performed using Caseless Matching as defined in section 3.13 of **[Unicode]**, with the exception that it is not necessary to perform normalization after case folding.
3. *IMPORTANT:* If there is no match, this comparison MUST be repeated after enclosing the value of the Path String parameter in parentheses (“(“ and “)”). This eliminates the

need for XRD authors to specify multiple `xrd:Path` elements in order to match an XRI path that may or may not be expressed as a cross-reference.

4. *IMPORTANT:* If there is still no match, the final subsegment of the Path String parameter and its preceding delimiter MUST be removed and the comparision process repeated beginning with step 1. This process MUST be repeated until the Path String is null. This enables stem-based path matching.

8.3 Selection Rules

After the matching rules in section 8.2 are applied, final selection of `xrd:Service` elements is governed by the value of the `xrd:select` attribute of each matched child element according to the following rules.

1. If the `xrd:match` attribute of any child element of an `xrd:Service` element has a value of `none`, then the `xrd:Service` element MUST NOT be selected, regardless of the value of any sibling selection element.
2. With the exception of the first rule above, If the `xrd:select` attribute of any matched child element of an `xrd:Service` element has a value of `true`, then the `xrd:Service` element MUST be selected (i.e., this is an “or” match).
3. If the `xrd:select` attribute for all matched child elements of an `xrd:Service` element has a value of `false` or null (the implied value if the attribute is absent), then the `xrd:Service` element MUST only be selected if there is a match on at least one `xrd:Type` child element, at least one `xrd:MediaType` child element, and at least one `xrd:Path` child element (i.e., this is an “and” match).

8.4 Construction of Service Endpoint URIs

If the output is a URI List, the final step in the service endpoint selection process is construction of the service endpoint URI(s). This is governed by the `xrd:append` attribute of each `xrd:URI` element. The values of this attribute are shown in Table 21.

Value	Component of QXRI to Append
none	None. This is the default for the authority resolution phase. (URI construction for authority resolution service endpoints is defined in section 5.1.7).
local	The entire local part, i.e., one of three cases: a) If only a path is present, append the path (including the leading “/”); b) If only a query is present, append the query (including the leading “?”) c) If both a path and a query are present, append the entire local part (including the leading “/”). This is the default for the service endpoint selection phase.
authority	Authority segment only (including the community root subsegment but no leading delimiter or “xri://”).
path	Path only (including the leading “/”).
query	Query only (including the leading “?”).
qxri	Entire QXRI (including the leading “xri://”).

Table 21: Values of the append attribute and the corresponding QXRI component to append.

If the `xrd:append` attribute is empty or omitted, the default value is `none` for the authority resolution phase and `local` for the service endpoint selection phase. Following are the rules for construction of the final service endpoint URI based on the value of the `xrd:append` attribute. *Note that these rules must be followed closely in order to give XRD 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 of the QXRI component specified in Table 21, *including any leading delimiter(s),* with no 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 21 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 21 is null, then the contents of the `xrd:URI` element MUST be returned directly as if the value of the `xrd:append` attribute was `none`.
3. If any special query parameters for XRI proxy resolution were added to an existing QXRI query component as defined in section 7.3, these query parameters MUST be removed prior to performing the append operation. 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.

9 Reference Processing

9.1 Synonyms

XRI resolution includes support for *synonyms*—XRIs that are not character-for-character equivalent, but which identify the same target resource. Three types of synonyms may be expressed in XRDs:

- *Local synonyms* are expressed using the `xrd:LocalID` element. A local synonym is an XRI that is interchangeable with the contents of the `xrd:Query` element. In other words, it is an XRI assigned by the same authority and identifing the same target resource as the current XRD (this authority is identified by the `xrd:XRD/xrd:ProviderID` element of the XRD if present—see section 3.2.) A common example is a persistent XRI assigned to a resource that has one or more reassignable XRIs. Resolution of a local synonym MUST return the same XRD as the one in which the local synonym appears (except for the value of the `xrd:Query`, `xrd:Expires`, and `xrd:LocalID` elements).
- *Canonical synonyms* are expressed using the `xrd:CanonicalID` element. A canonical synonym is the preferred absolute XRI among all absolute XRI synonyms for a resource (in the context of the authority providing the current XRD). This XRI may or may not be assigned by the same authority providing the current XRD, and if resolvable may or may not resolve to the current XRD. For durability, a canonical synonym SHOULD be a persistent XRI. For simplicity, only one canonical synonym SHOULD be assigned to a resource, however in certain circumstances (such as merging two previously distinct resources into one), this may not be possible.
- *References* are expressed using the `xrd:Ref` element. In XRI resolution, a reference is an absolute XRI that identifies the same target resource as the query XRI but which MUST be assigned by a different authority than the authority providing the current XRD. A reference MAY resolve to a different XRD than the current XRD, i.e., an XRD containing different synonyms, service endpoints, or other metadata describing the target resource. (Note that such an XRD MAY also include a reference back to the current QXRI, in which case the references will be circular.)

Of these three synonym types, references play a special role in XRI resolution. In both authority resolution and service endpoint selection, if the current XRD does not contain a requested `xrd:Service` element but contains at least one `xrd:Ref` element, a resolver may follow these reference(s) according to the rules defined in this section.

9.2 Processing Rules

Figure 4 is an overview of the logical flow of selecting a reference for processing.

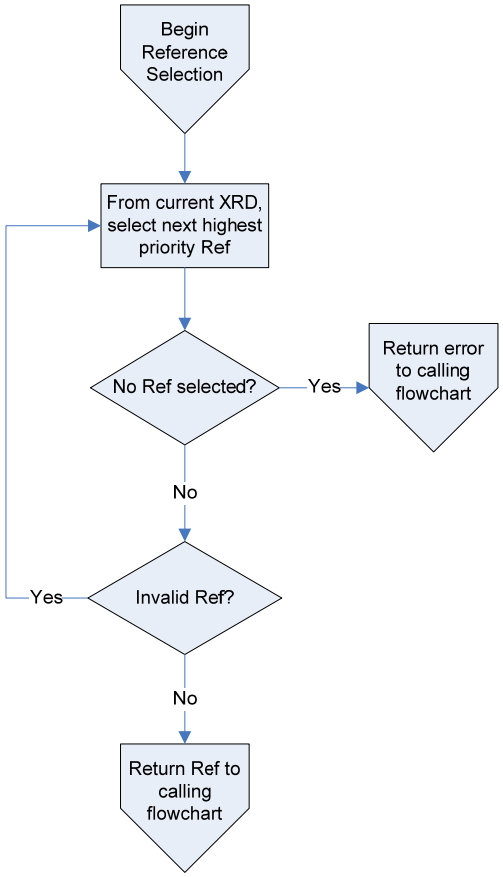

Figure 4: Flowchart for selecting an XRI reference for processing.

Following are the normative rules that apply to reference processing:

1. The resolver MUST begin by selecting the highest priority `xrd:Ref` element in the current XRD. If not found, this is an error and the resolver MUST proceed as defined in section 5 for authority resolution or section 8 for service endpoint selection.
2. The contents of the selected `xrd:Ref` element MUST be a valid QXRI as defined in section 4.1.1. If not, it MUST be ignored and the step above repeated to select the next highest priority reference.
3. Once a valid reference has been selected, the resolver MUST begin resolution of a new XRDS document beginning with the community root authority of the reference XRI as defined in section 5.1.3. The resolver MUST use the same resolu-tion input parameters as for the original QXRI. For reference processing to complete successfully, the resolver MUST complete resolution of the entire Authority String of the reference XRI (including following any further references if necessary). If the reference XRI includes a Path String or Query String (including any resolution query parameters), they MUST be ignored.
4. If reference processing is successful and the Resolution Media Type is an XRDS document, the XRDS document resulting from reference processing MUST be nested inside the parent XRDS document as defined in section 9.3.

5. If reference processing is successful and the Resolution Media Type is an XRD document, the output MUST be the final XRD document returned.
6. If reference processing is successful and the Resolution Media Type is a URI List or an HTTP(S) redirect, it MUST be based on the final XRD document returned.

9.3 Nesting XRDS Documents

If a reference is followed successfully, it will produce an XRDS document that fully describes the reference. This XRDS document MUST be included in the containing XRDS document immediately following the `xrd:XRD` element that contains the `xrd:Ref` element being followed. In addition, the `xrds:XRDS/@xrds:ref` attribute of this nested XRDS document MUST be set to the value of the `xrd:Ref` element it describes.

This allows a consuming application to verify the complete chain of XRDs obtained to resolve the original QXRI even if resolution traverses references. Note that nested XRDS documents do 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 whose `xml:id` attribute value matches 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 `xrd:idref` attribute value to the value of the `xrd:XRD/xml:id` attribute of the matched XRD element. This prevents conflicting `xrd:XRD/xml:id` values.

In the following example the original query XRI is `xri://@a*b*c`. The XRD for `xri://@a*b` does not contain an authority resolution service endpoint but includes a reference to `xri://@x*y`. The elements and attributes specific to reference processing are shown in bold.

```
<XRDS xmlns="xri://$xrds" ref="xri://@a*b*c">
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
        <Query>*a</Query>
        ...
        <Service>
              <Type>xri://$res*auth*($v*2.0)</Type>
              <URI>http://a.example.com</URI>
        </Service>
  </XRD>
  <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
        <Query>*b</Query>
        ...
        <Ref>xri://@x*y</Ref>
        <Service>
        ...no authority resolution service endpoint...
        </Service>
  </XRD>
  <XRDS ref="xri://@x*y">
        <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
              <Query>*x</Query>
              ...
              <Service>
                    <Type>xri://$res*auth*($v*2.0)</Type>
                    <URI>http://x.example.com</URI>
              </Service>
        </XRD>
        <XRD xmlns="xri://$xrd*($v*2.0)" version="2.0">
              <Query>*y</Query>
              ...
              <Service>
                    <Type>xri://$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>
        ...
        <Service>
        ...final service endpoints described here...
        </Service>
  </XRD>
</XRDS>
```

10 Error Processing

10.1 Error Codes

XRI resolution error 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.

Each major category is broken into five 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.

The full list of XRI resolution error codes is defined in Table 22.

Code	Symbolic Error	Phase(s)	Description
100	SUCCESS	Any	Operation was successful.
101	REF_NOT_FOLLOWED	Any	Operation was successful to the point where a reference needed to be processed, but the resolver was instructed by the `refs` parameter not to follow references.
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_RES_MEDIA_TYPE	Input	Input Resolution Media Type 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 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.
230	TRUSTED_RES_ERROR	Trusted resolution	Generic trusted resolution error
231	HTTPS_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS authority resolution endpoint.
232	SAML_RES_NOT_FOUND	Trusted resolution	The resolver was unable to locate a SAML authority resolution endpoint.
233	HTTPS+SAML_RES_ NOT_FOUND	Trusted resolution	The resolver was unable to locate an HTTPS+SAML authority resolution endpoint.
234	UNVERIFIED_SIGNATURE	Trusted resolution	Signature verification failed.
240	SEP_SELECTION_ERROR	SEP selection	Generic service endpoint selection error
241	SEP_NOT_FOUND	SEP selection	The requested service endpoint could not be found in the current XRD or via reference processing.
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 resolution service, the server returned a non-200 HTTP status.
322	INVALID_XRDS	Authority resolution	Invalid XRDS received from an authority resolution service (includes malformed XML, truncated content, or wrong content type).

1567 **Table 22: Error codes for XRI resolution.**

As defined in section 4.2, when resolution output is an error and the output format is an XRDS Document or XRD Document, the error code is returned as the value of the `xrd:code` attribute of the `xrd:Status` element. When the resolution output is a URI List, the error code is returned as the first line of a plain text message.

10.2 Error Context Strings

Each error code in Table 22 may be returned with an optional error context string that provides additional human-readable information about the error. When resolution output is an XRDS Document or XRD Document, this string is returned as the contents of the `xrd:Status` element. When the resolution output is a URI List, this string is returned as the second line of a plain text message. 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.

10.3 Error Handling in Recursing and Proxy Resolution

In recursing and proxy resolution (sections 5.1.5 and 7), 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 Media Type as defined in section 4.2.

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. 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 Document, it MUST include the `xrd:Query` element that produced the error and the `xrd:Status` element that describes the error as defined above.

If this output format is a URI List, it MUST be returned with the content type `text/plain`. The first line MUST consist of only the numeric error code as defined in section 10.1 followed by a CRLF. The second line is OPTIONAL; if present it MUST be the error context string as defined in section 10.2.

If the value of the Resolution Media Type is null (which can only happen in proxy resolution as described in section 7.5), rather than returning an HTTP(S) redirect, a proxy resolver SHOULD return a human-readable error message with a media type of either `text/plain` or `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.

11 Use of HTTP(S)

11.1 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 10. In this way calling applications do not have to deal separately with XRI and HTTP error messages.

11.2 HTTP Headers

11.2.1 Caching

The HTTP caching capabilities described by **[RFC2616]** should be leveraged for all types of XRI resolution service. 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.

11.2.2 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 11.2.1.

11.2.3 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.

11.3 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.

11.4 Caching and Efficiency

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 identifier 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.

12 Extensibility and Versioning

12.1 Extensibility

12.1.1 Extensibility of XRDs

The XRD schema in Appendix A 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 XRD document.

Extension specifications MAY simulate “Must Understand” behavior by applying an “enclosure” pattern. Elements defined by the XRD schema in Appendix A 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>
      …
      <other:ExtensionElement>…</other:ExtensionElement>
    </Service>
  </other:SuperService>
</XRD>
```

In this example, the `other:ExtensionElement` modifies the interpretation or processing rules for the parent `xrd:Service` element and therefore must be understood by the consumer for the proper interpretation of the parent `xrd:Service` element. To preserve the correct interpretation of the `xrd:Service` element in this context, the `xrd:Service` element is “wrapped” 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.

12.1.2 Other Points of Extensibility

The use of HTTP, 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 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 **[XRIMetadata]**.

12.2 Versioning

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

12.2.1 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:

$$Major_B > Major_A \text{ OR } ((Major_B = Major_A) \text{ AND } Minor_B > Minor_A)$$

12.2.2 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.

12.2.3 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 element, and can remain stable indefinitely because there is no need to version its namespace.

The `xrd:XRD` element has an optional `xrd:version` attribute. 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.

With regard to versioning, this specification follows the same 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.

12.2.4 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 2.1.2.

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.

13 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.

13.1 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.

13.2 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

13.3 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.

13.4 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.

13.5 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.

13.6 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.

13.7 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.

13.8 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.

14 References

14.1 Normative

[DNSSEC] D. Eastlake, *Domain Name System Security Extensions*, http://www.ietf.org/rfc/rfc2535, 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, 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, RFC 2046, November 1996.

[RFC2119] S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, RFC 2119, March 1997.

[RFC2141] R. Moats, *URN Syntax*, http://www.ietf.org/rfc/rfc2141.txt, IETF RFC 2141, May 1997.

[RFC2234] D. H. Crocker and P. Overell, *Augmented BNF for Syntax Specifications: ABNF*, http://www.ietf.org/rfc/rfc2234.txt, RFC 2234, November 1997.

[RFC2483] M. Meallling, R. Daniel Jr., *URI Resolution Services Necessary for URN Resolution,* http://www.ietf.org/rfc/rfc2483.txt, 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, RFC 2616, June 1999.

[RFC2818] E. Rescorla, *HTTP over TLS*, http://www.ietf.org/rfc/rfc2818.txt, RFC 2818, May 2000.

[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.

14.2 Informative

[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.

Appendix A. XML Schema for XRDS and XRD (Normative)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="xri://$xrds" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xrds="xri://$xrds">
    <!-- 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"/>
            <xs:attribute name="ref" type="xs:anyURI" use="optional"/>
        </xs:complexType>
    </xs:element>
</xs:schema>


<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="xri://$xrd*($v*2.0)" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xrd="xri://$xrd*($v*2.0)">
    <!-- 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="selectionAttrGrp">
        <xs:attribute name="match" use="optional" default="default">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="default"/>
                    <xs:enumeration value="content"/>
                    <xs:enumeration value="any"/>
                    <xs:enumeration value="non-null"/>
                    <xs:enumeration value="null"/>
                    <xs:enumeration value="none"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="select" type="xs:boolean" use="optional" default="false"/>
    </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="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:Expires" minOccurs="0"/>
                <xs:element ref="xrd:ProviderID" minOccurs="0"/>
                <xs:element ref="xrd:LocalID" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="xrd:CanonicalID" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="xrd:Ref" 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:attribute name="code" type="xs:int" use="required"/>
                    <xs:attributeGroup ref="xrd:otherattribute"/>
                </xs:extension>
            </xs:simpleContent>
        </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="CanonicalID" type="xrd:URIPriorityPattern"/>
    <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:element ref="xrd:URI" 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">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xrd:URIPattern">
                    <xs:attributeGroup ref="xrd:priorityAttrGrp"/>
                    <xs:attribute name="append">
                        <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:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Appendix B. RelaxNG Compact Syntax Schema for XRDS and XRD (Informative)

[TODO]

Appendix C. Media Type Definition for application/xrds+xml (Normative)

[TODO]

Appendix D. Media Type Definition for application/xrd+xml (Normative)

[TODO]

Appendix E. Example Local Resolver Interface Definition (Informative)

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

The interface definition is provided as five operations where each operation takes three or more of the following input parameters. The input parameters are described here in terms of the normative text in section 4. 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 4.1.1.
`trustType`	The value of the `trust` media type subparameter of the Resolution Media Type parameter as specified in Table 6 of section 2.3, whose behavior is defined in section 4.1.2.
`followRefs`	The value of the `refs` media type subparameter of the Resolution Media Type parameter as specified in Table 6 of section 2.3, whose behavior is defined in section 4.1.2.
`sepType`	Service Type as defined in Table 8 of section 4.1.
`sepMediaType`	Service Media Type as defined in Table 8 of section 4.1.

The five operations correspond to the following combinations of values of the Resolution Media Type parameter and its `sep` (service endpoint) subparameter (section 4.1.2) as shown below.

	Operation name	Resolution Media Type 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

```
int resolveAuthToXRDS(
        in string QXRI, in string trustType, in boolean followRefs,
        out string XRDS, out string errorContext);
```

- Performs authority resolution only (sections 5 and 6) and outputs the XRDS as specified in section 4.2.1 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.
- The output XRDS argument will be signed or not depending on the value of `trustType`.
- Returns the error code. If error, then the errorContext output argument may contain additional error information. The output XRDS will contain a final XRD with the same status code and optional context information in its `xrd:Status` element.

2. Resolve Authority to XRD

```
int resolveAuthToXRD(
        in string QXRI, in string trustType, in boolean followRefs,
        out string XRD, out string errorContext);
```

- Performs authority resolution only (sections 5 and 6) and outputs the final XRD as specified in section 4.2.2 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.
- The output XRD argument will be signed or not depending on the value of `trustType`.
- Returns the error code. If error, then the `errorContext` output argument may contain additional error information. The output XRD will contain the same status code and optional context information in its `xrd:Status` element.

3. Resolve Service Endpoint to XRDS

```
int resolveSEPToXRDS(
        in string QXRI, in string trustType, in string sepType,
        in string sepMediaType, in boolean followRefs,
        out string XRDS, out string errorContext);
```

- Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8) and outputs the XRDS as specified in section 4.2.1 when the `sep` subparameter is `true`.
- As specified in section 4.2.1, the output XRDS document will contain a nested XRDS document as the final child element if reference processing was necessary to locate the request service endpoint and the `followRefs` flag was set to `true`.
- The final XRD in the output XRD will either contain at least one instance of the requested service endpoint or an error.
- The output XRDS argument will be signed or not depending on the value of `trustType`.
- Returns the error code. If error, then the `errorContext` output argument may contain additional error information. The output XRDS will contain a final XRD with the same status code and optional context information in its `xrd:Status` element.
- For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the same as the value null.

4. Resolve Service Endpoint to XRD

```
int resolveSEPToXRD(
        in string QXRI, in string trustType, in string sepType,
        in string sepMediaType, in boolean followRefs,
        out string XRDS, out string errorContext);
```

- Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8) and outputs the XRD as specified in section 4.2.2 when the `sep` subparameter is `true`.
- As specified in section 4.2.2, all elements in the output XRD subject to the global `xrd:priority` attribute will be returned in order of highest to lowest.
- The output XRD will either contain at least one instance of the requested service endpoint or an error.
- The output XRD will be *not* be signed regardless of the value of `trustType` because the XRD will be filtered to only the selected service endpoints.
- Returns the error code. If error, then the `errorContext` output argument may contain additional error information. The output XRD will contain the same status code and optional context information in its `xrd:Status` element.
- For parameters `sepType` and `sepMediaType`, the value empty string ("") is interpreted the same as the value null.

5. Resolve Service Endpoint to URI List

```
int resolveSepToURIList(
        in string QXRI, in string trustType, in string sepType,
        in string sepMediaType, in boolean followRefs,
        out string[] URIList, out string errorContext);
```

- Performs authority resolution (sections 5 and 6) and service endpoint selection (section 8) and, upon success, outputs a non-empty URI List as specified in section 4.2.3.
- Returns the error code. If error, then the output URI List will be empty, and the `errorContext` output argument may contain additional error information.
- For parameters `sepType` and `sepMediaType`, the value empty string (“”) is interpreted the same as the value null.

Appendix F. Examples of Generic Authority Resolution (Informative)

[TODO]

Appendix G. Examples of SAML Trusted Authority Resolution (Informative)

[TODO]

Appendix H. Examples of Service Endpoint Selection (Informative)

[TODO]

Appendix I. Acknowledgments

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

- Geoffrey Strongin, Advanced Micro Devices
- Ajay Madhok, AmSoft Systems
- Jean-Jacques Dubray, Attachmate
- William Barnhill, Booz Allen and Hamilton
- Drummond Reed, Cordance Corporation
- Marc Le Maitre, Cordance Corporation
- Dave McAlpin, Epok
- Loren West, Epok
- Peter Davis, NeuStar
- Masaki Nishitani, Nomura Research
- Nat Sakimura, Nomura Research
- Tetsu Watanabe, Nomura Research
- Owen Davis, PlaNetwork
- Victor Grey, PlaNetwork
- Fen Labalme, PlaNetwork
- Mike Lindelsee, Visa International
- Gabriel Wachob, Visa International
- Dave Wentker, Visa International
- Bill Washburn, XDI.ORG

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

Thomas Bikeev, EAN International; Krishna Sankar, Cisco; Winston Bumpus, Dell; Joseph Moeller, EDS; Steve Green, Epok; Lance Hood, Epok; Adarbad Master, Epok; Davis McPherson, Epok; Chetan Sabnis, Epok; Phillipe LeBlanc, GemPlus; Jim Schreckengast, Gemplus; Xavier Serret, Gemplus; John McGarvey, IBM; Reva Modi, Infosys; Krishnan Rajagopalan, Novell; Tomonori Seki, NRI; James Bryce Clark, OASIS; Marc Stephenson, TSO; Mike Mealling, Verisign; Rajeev Maria, Visa International; Terence Spielman, Visa International; John Veizades, Visa International; Lark Allen, Wave Systems; Michael Willett, Wave Systems; Matthew Dovey; Eamonn Neylon; Mary Nishikawa; Lars Marius Garshol; Norman Paskin; Bernard Vatant.

- A special acknowledgement to Jerry Kindall (Epok) for a full editorial review.

Appendix J. Notices

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's procedures with respect to rights in OASIS specifications can be found at 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 implementors or users of this specification, can be obtained from the OASIS President.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS President.